ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA DE GESTIÓN DE DATOS PARA EL MONITOREO INTEGRAL DE GLACIARES

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

JAVIER ALEJANDRO ARMAS NAVARRETE

javier.armas@epn.edu.ec

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, MSc.

david.mejia@epn.edu.ec

CODIRECTOR: ING. MARCOS JOSHUA VILLACÍS ERAZO, PhD.

marcos.villacis@epn.edu.ec

Quito, julio 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Javier Alejandro Armas Navarrete, bajo nuestra supervisión.

ING. RAÚL DAVID MEJÍA NAVARRETE, MSc
DIRECTOR DEL TRABAJO DE TITULACIÓN

ING. MARCOS JOSHUA VILLACÍS ERAZO, PhD
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Javier Alejandro Armas Navarrete, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

JAVIER ALEJANDRO ARMAS NAVARRETE

DEDICATORIA

A mi familia, porque sin su apoyo no podría haber llegado al lugar en el que estoy. Por su entrega, amor y paciencia. Esto es para ustedes.

A mí, porque he sabido superar cada obstáculo que la vida me ha puesto, y, a pesar de las dificultades, nunca he claudicado en mis objetivos.

AGRADECIMIENTO

A mi padre, por siempre brindarme su apoyo y no permitir que pierda el rumbo. Por enseñarme que con esfuerzo y perseverancia se puede lograr todo.

A mi madre, por su amor incondicional y por ser un refugio seguro en tiempos difíciles. Por ser fuente de cariño, bondad y calma de una manera tan particular.

A mi hermana, por siempre orar por mí. Por entender y perdonar mis errores, y por enseñarme otra faceta del mundo que mi vida necesita para estar en equilibrio.

A mis amigos, por siempre estar ahí. Por tantas risas y tristezas que nos convirtieron en mucho más que compañeros.

Al Ing. David Mejía, por su paciencia y acertada guía en el desarrollo de este Trabajo de Titulación.

Al Ing. Marcos Villacís, por haberme permitido formar parte de su proyecto de investigación y desarrollar mi Trabajo de Titulación.

A todos aquellos profesores de la Escuela Politécnica Nacional que se convirtieron no solo en educadores, sino también en ejemplos de personas.

A la vida, por tantas enseñanzas que, aunque no siempre fueron fáciles de comprender, dejaron huella en mí, permitiéndome crecer y buscar siempre mi mejor versión.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El presente Proyecto Técnico consiste en el desarrollo de un sistema de gestión de datos para el monitoreo integral de glaciares. El sistema consta de dos componentes: una base de datos y una aplicación web.

Este documento está organizado en cuatro capítulos. En el primer capítulo se detallan las tecnologías implicadas en el desarrollo de este sistema, iniciando con una descripción de la arquitectura REST, el *framework* ASP.NET MVC, bases de datos relacionales, Entity Framework y series temporales. Asimismo, se explica la creación de mapas interactivos mediante Leaflet y GeoJSON. Finalmente, se describe la metodología ágil Kanban, usada durante el desarrollo del sistema propuesto.

En el segundo capítulo se detallan los requerimientos funcionales y no funcionales del sistema. Se explica también el proceso de diseño de los componentes del sistema, empezando con la base de datos y luego la aplicación web. Posteriormente se describe el proceso de implementación de ambos componentes.

El tercer capítulo consta de las pruebas de funcionamiento realizadas al concluir el sistema. También están las entrevistas de validación aplicadas a los miembros del proyecto LMI GREAT ICE. Finalmente se tiene un resumen de los errores encontrados durante las pruebas realizadas y su respectiva corrección.

El cuarto capítulo contiene las conclusiones y recomendaciones elaboradas al finalizar el proyecto.

Adicionalmente, se presentan anexos que contienen los resultados de las entrevistas aplicadas, el código de creación y población de la base de datos, el código de la aplicación web, los resultados de las entrevistas de validación y el manual de instalación y uso.

PALABRAS CLAVE: MVC, Entity framework, mapas interactivos, información glaciológica, tareas asíncronas, gráficas temporales.

ABSTRACT

The current Technical Project consists of the development of a data management system for the integral monitoring of glaciers. The system consists of two components: a database and a web application.

This document is organized in four chapters. The first chapter details the technologies involved in the development of this system, starting with a description of the REST architecture, the ASP.NET MVC framework, relational databases, Entity Framework and time series. Also, the creation of interactive maps through Leaflet and GeoJSON is explained. Finally, Kanban agile methodology used during the development of the proposed system is described.

In the second chapter, the functional and non-functional requirements of the system are detailed. The process of designing the components of the system is also explained, starting with the database and then the web application. Subsequently, the process of implementing both components is described.

The third chapter consists of the functional tests carried out at the conclusion of the system. There are also validation interviews applied to members of the LMI GREAT ICE project. Finally, we have a summary of the errors found during the tests and their respective correction.

The fourth chapter contains the conclusions and recommendations prepared at the end of the project.

Additionally, appendixes contain results of the interviews applied, code for create and populate the database, code of the web application, results of the validation interviews, and the installation and use manual are presented.

KEYWORDS: MVC, Entity framework, interactive maps, glaciological data, asynchronous tasks, temporary graphs.

1. INTRODUCCIÓN

El objetivo de este Trabajo de Titulación es desarrollar un sistema de gestión de datos para el monitoreo integral de glaciares. Este sistema consta de dos componentes, una base de datos y una aplicación web.

La base de datos, elaborada en SQL SERVER 2017, permite el almacenamiento estructurado de información.

La aplicación web está desarrollada en ASP.NET MVC 5, y permite la gestión de datos glaciológicos y la visualización de los mismos. El acceso a los datos se realiza a través de Entity Framework – *Database First*. Además, la aplicación web es capaz de leer archivos de Meteorología, Glaciología e Hidrología que cumplan con un formato establecido, almacenar su información en la base de datos y guardar los archivos originales en el servidor. También se permite la búsqueda de información con distintos criterios y la presentación de resultados mediante gráficas temporales. Los resultados de la búsqueda están disponibles para su descarga en formato CSV (*comma-separated values*). Asimismo, se cuenta con un mapa elaborado utilizando la librería Leaflet y GeoJSON para la representación de datos geográficos. En el mapa se muestran los marcadores de las estaciones en los glaciares Antisana (Ecuador) y Zongo (Bolivia) que son estudiados actualmente por el proyecto LMI GREAT ICE. La aplicación web, además permite la realización de CRUD (*Create-Read-Update-Delete*) de distintas entidades. Se cuenta también con tres perfiles de usuario: `Administrator`, `Download&Upload` y `Download`, cada uno con distintos privilegios.

1.1 OBJETIVOS

El objetivo general de este Trabajo de Titulación es desarrollar un sistema de gestión de datos para el monitoreo integral de glaciares.

Los objetivos específicos de este Trabajo de Titulación son:

- Analizar las tecnologías de software necesarias para el desarrollo de aplicaciones web.
- Diseñar los componentes que el sistema tendrá para cumplir con los requerimientos solicitados.
- Implementar los componentes con base en el diseño realizado.

1.2 ALCANCE

Este Trabajo de Titulación busca elaborar un sistema que permita gestionar y presentar datos glaciológicos del Ecuador. El sistema estará conformado por dos componentes, una base de datos y una aplicación web. Se muestra una descripción general del sistema en la Figura 1.1.

Para el desarrollo de este sistema se empleará la metodología Kanban.

Para determinar los requerimientos funcionales y no funcionales del sistema se realizarán y aplicarán 5 entrevistas al personal de LMI GREAT ICE.

Se establecerá el diagrama entidad-relación de la base de datos en la que se almacenará la información de los glaciares provista por el proyecto LMI GREAT ICE. Existen casos en que, por fallas de los sensores en las estaciones, la información de las mediciones es inconsistente, por lo que se realizará un procesamiento y depuración de dicha información previo a su almacenamiento. Como parte del procesamiento y depuración se definirá una estructura fija que deben cumplir los documentos para subirlos al sistema. También, se revisará la información de los documentos, se determinarán aquellas celdas que no cumplan con el formato establecido y se reemplazará su valor por otro establecido en la etapa de diseño.

La aplicación web será implementada utilizando ASP.NET MVC 5. La aplicación web permitirá presentar y gestionar la información almacenada en la base de datos. Se utilizarán series temporales que permitan visualizar información como temperatura, velocidad del viento, albedo, precipitaciones, humedad, entre otras que se definan en la etapa de diseño.

También, se definirán 3 perfiles de usuario: `Administrator`, `Download&Upload` y `Download`, con distintos privilegios.

Cada perfil determinará el nivel de acceso a la información que tiene cada usuario. Adicionalmente, se contará con un mapa en el que se mostrarán las estaciones de medición en los glaciares, y, al dar clic sobre alguna, se presentará la información correspondiente a la estación seleccionada. Se presentará al menos la información de las estaciones ubicadas en el volcán Antisana, y que se encuentran dentro y fuera del glaciar `ANTISANA-15`.

Otra manera para la presentación de datos será a través de filtros de búsqueda.

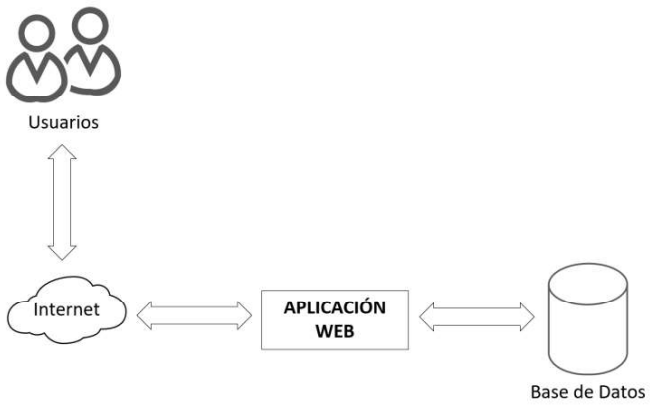


Figura 1.1 Descripción de la arquitectura del sistema

1.3 MARCO TEÓRICO

1.3.1 REPRESENTATIONAL STATE TRANSFER (REST)

REST es una arquitectura de software que provee estándares para facilitar la comunicación entre sistemas en la web [1].

REST está basado en un conjunto de principios que definen cómo se describen y se acceden a recursos en la red [2]. Los sistemas que cumplen este conjunto de principios se denominan RESTful.

A continuación, se enumeran los principios esenciales para que un sistema sea considerado RESTful [3]:

- **Basado en cliente-servidor:** Al separar la lógica de la aplicación de la interfaz de usuario se mejora la portabilidad entre plataformas y la escalabilidad.
- **Sin estado (*Stateless*):** El cliente debe proveer toda la información necesaria para que la aplicación en el servidor ejecute alguna acción.

  Cada solicitud del cliente debe contener todo lo necesario para que dicha solicitud sea interpretada por el sistema.

  No se puede aprovechar ningún contexto almacenado en el servidor, por lo que el estado de una sesión se mantiene solo en el lado del cliente.

- ***Cacheable*:** La respuesta a una solicitud debería poder almacenarse en la memoria caché del cliente.

  De esta forma, el cliente podría reutilizar esos datos de la respuesta en solicitudes similares posteriores. Esto permite disminuir el tiempo de respuesta ya que no es necesario enviar nuevamente la solicitud al servidor.

- **Interfaz uniforme:** Se deben cumplir algunas restricciones que guíen el comportamiento de los componentes para tener una interfaz uniforme, en el caso de REST, estas restricciones son la identificación de recursos y la manipulación de recursos a través de representaciones.

  REST está basada en el protocolo HTTP[1], por lo que el acceso a los recursos se hace a través de los *Uniform Resource Identifier* (URI)[2] [4] .

  Los recursos no pueden ser directamente accedidos o modificados, sino que se trabaja con representaciones de ellos [5]. La representación de recursos en REST puede hacerse en varios formatos, como XML[3] y JSON[4].

  JSON es muy utilizado puesto que es un formato ligero y sencillo que permite el intercambio de datos independientemente del lenguaje de programación [6].

  Existen dos estructuras básicas en JSON: un conjunto de pares nombre/valor no ordenados, que pueden representar objetos; y, una lista ordenada de valores, que puede representar un arreglo. Sumado a estas dos estructuras básicas, se admiten también datos de tipo: `string`, `number`, `bool` y `null` [7].

  En el Código 1.1 se presenta un ejemplo de objeto JSON. La línea 3 indica el nombre del objeto, mientras que en el resto de líneas se especifican los pares nombre/valor de los atributos del objeto. Cada par nombre/valor debe estar separado por una coma.

- **Sistema en capas:** Restringe el comportamiento de los componentes para que cada componente no pueda ver más allá de la capa con la que está interactuando, dentro de un sistema de capas jerárquico.

---

[1] HTTP (*Hypertext Transfer Protocol*): Es un protocolo que permite la transferencia de información en la *World Wide Web* y actúa en la capa de aplicación.
[2] URI (*Uniform Resource Identifier*): Es un conjunto de caracteres que identifican un recurso en particular sin ninguna ambigüedad.
[3] XML (*eXtensible Markup Language*): Es un lenguaje de marcado que define un conjunto de reglas para codificar datos en un formato que puede ser leído por humanos y máquinas. XML fue diseñado para almacenar y transportar datos.
[4] JSON (*JavaScript Object Notation*): Es un formato de intercambio y almacenamiento de datos estandarizado. Los datos en JSON son transmitidos como texto.

```
{
 “usuario”: {
    “idUsuario”: 1,
    “nombre”: Alejandro,
    “apellido”: Navarrete
    }
}
```

Código 1.1 Ejemplo de objeto JSON

- **Código bajo demanda (opcional):** Permite extender la funcionalidad del cliente a través de la descarga y ejecución de código en forma de *scripts* o *applets*.

Los sistemas RESTful, entonces, tienen las siguientes ventajas: escalabilidad, alto rendimiento, sencillez en la implementación, son livianos y, al utilizar HTTP, prácticamente pueden ser consumidos por cualquier aplicación o dispositivo [4].

1.3.2 ASP.NET MVC

ASP.NET MVC es un *framework*[5] elaborado por Microsoft para el desarrollo de aplicaciones web que implementa el patrón MVC (Modelo-Vista-Controlador).

El *framework* es de código abierto, gratuito y viene integrado en Visual Studio 2017. ASP.NET MVC permite la construcción de sitios web mediante HTML[6], CSS[7] y JavaScript[8]. El código de programación puede ser implementado utilizando lenguajes como C# o VB.NET.

MVC es un patrón de arquitectura de software que permite desarrollar una aplicación de manera modular. Esto implica que, al modificar el código de uno de los módulos, no se

[5] Framework: Es una infraestructura que proporciona un conjunto de clases y librerías que se reutilizan para la creación de elementos recurrentes y que definen una arquitectura para el desarrollo de software.
[6] HTML (*Hypertext Markup Language):* Es un lenguaje de marcado que se utiliza en la creación de páginas web y aplicaciones web. HTML utiliza etiquetas (*tags*) para describir la estructura de la página o aplicación web que se está creando.
[7] CSS (*Cascading Style Sheets*): Es un lenguaje de hojas de estilo que describe cómo se van a presentar los elementos HTML en la pantalla. Se usa para dar estilo a la apariencia del contenido, por ejemplo, fuente de letras o colores.
[8] JavaScript: Es un lenguaje de programación de alto nivel muy utilizado en el ámbito web, puesto que permite el desarrollo de sitios web interactivos. Conjuntamente con HTML y CSS forman el núcleo de tecnologías esenciales en la *World Wide Web*.

afecta el comportamiento de los otros. De esta manera se cumple con el Principio de Responsabilidad Única (SRP)[9].

La idea general detrás del patrón MVC es que cada sección del código tiene un propósito. MVC permite separar los componentes de una aplicación dependiendo de la responsabilidad que tienen. Es decir, a través de MVC se puede organizar el código de mejor manera para que la aplicación sea flexible y escalable.

El funcionamiento del patrón MVC se presenta en la Figura 1.2. Inicialmente, el usuario realiza una petición al manipular la interfaz gráfica desde su computador, esto se denomina solicitud HTTP. En el lado del servidor, la petición es recibida por el Controlador, quien se encarga de consultar al Modelo los datos solicitados o realizar las acciones respectivas. Una vez hecho esto, el Controlador elige la Vista y le envía los datos correspondientes para actualizarla. La Vista genera una plantilla con la información respectiva, misma que es enviada de regreso al usuario. En el lado del cliente, el navegador interpreta la información y la muestra en la pantalla del usuario.

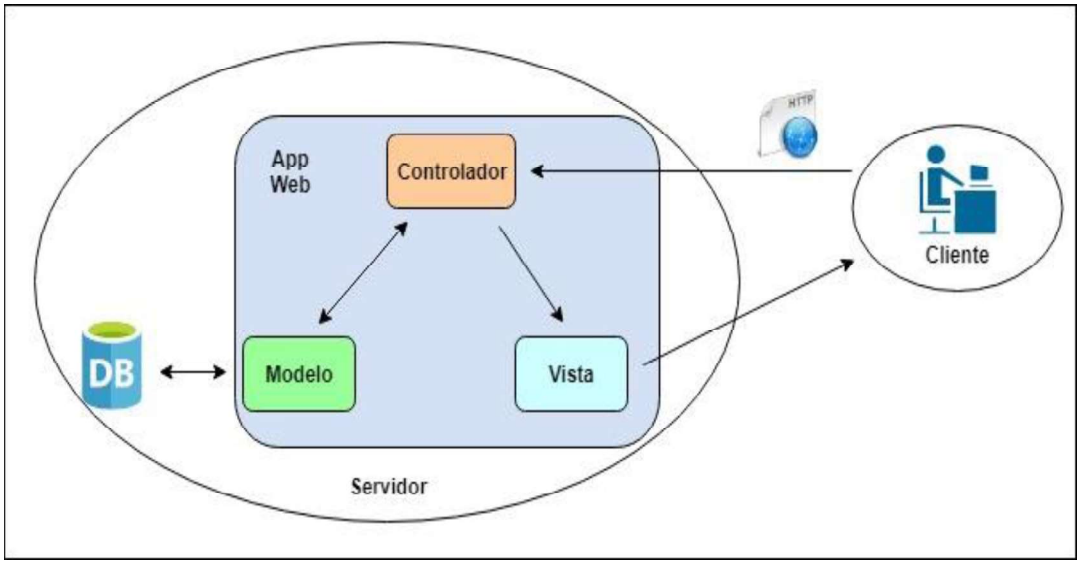


Figura 1.2 Funcionamiento del patrón Modelo-Vista-Controlador (MVC)

Los componentes de MVC son: Modelo, Vista y Controlador.

[9] SRP (*Single Responsibility Principle*): Es una primicia que establece que, cada clase o módulo tiene responsabilidad sobre una sola parte de la funcionalidad provista por el software.

1.3.2.1 Modelo

Define todo lo referido a los datos que el software utiliza. Los objetos del Modelo retornan información de la base de datos y almacenan información en la misma.

El Modelo es un conjunto de clases que describe los datos con los que se está trabajando y las reglas para cambiarlos y manipularlos [8].

El Modelo es totalmente independiente del Controlador y de la Vista. Para construir el Modelo, puede utilizarse Entity Framework[10].

ASP.NET MVC dispone de varias características y herramientas que facilitan y agilizan el desarrollo de aplicaciones web. Por lo general, estas herramientas crean plantillas genéricas, facilitan la interacción entre distintos lenguajes de programación u optimizan la codificación [9].

Entre las herramientas mencionadas están:

Scaffolding: Permite agregar rápidamente código que interactúa con el Modelo de datos mediante plantillas genéricas que reducen la cantidad de tiempo que se utiliza en desarrollar operaciones estándar.

En otras palabras, *Scaffolding* permite generar plantillas con código genérico a partir de un Modelo. El código generado corresponde a Controladores y a Vistas.

Visual Studio 2017 permite que, al crear un Controlador, la herramienta de *Scaffolding* genere código base de acciones como `Index`, `Details`, `Create`, `Edit` y `Delete`. El desarrollador únicamente debe modificar el código base de acuerdo con sus necesidades.

Un ejemplo de la plantilla generada mediante *Scaffolding* al añadir un Controlador se presenta en el Código 1.2. Por convención, el nombre del Controlador está dado por el nombre de la entidad del Modelo utilizada, en plural, seguido de la palabra `Controller`.

Las Vistas se crean con base en las acciones del Controlador que se deseen manipular.

Al agregar una Vista, se debe escoger la plantilla de la acción del Controlador asociada y el nombre de la clase del Modelo, tal como se muestra en la Figura 1.3. La Vista generada contiene código HTML que, de ser necesario, deberá ser modificado por el desarrollador.

[10] Entity Framework: Es una herramienta de Microsoft para el acceso a datos que permite asociar las tablas de una base de datos con clases y objetos.

Data Annotations: Son un mecanismo utilizado por ASP.NET MVC para validar el ingreso de datos del usuario.

Esta herramienta se implementa en las clases del Modelo y permite la validación del mismo en el lado del cliente.

```
0 referencias
public class gendersController : Controller
{
    // GET: genders
    0 referencias | 0 solicitudes | 0 excepciones
    public ActionResult Index()
    {
        return View();
    }

    // GET: genders/Details/5
    0 referencias | 0 solicitudes | 0 excepciones
    public ActionResult Details(int id)
    {
        return View();
    }

    // GET: genders/Create
    0 referencias | 0 solicitudes | 0 excepciones
    public ActionResult Create()
    {
        return View();
    }

    // POST: genders/Create
    [HttpPost]
    0 referencias | 0 solicitudes | 0 excepciones
    public ActionResult Create(FormCollection collection)
    {
        try
        {
            // TODO: Add insert logic here

            return RedirectToAction("Index");
        }
        catch
        {
            return View();
        }
    }

```

Código 1.2 Plantilla generada mediante *Scaffolding* al crear un Controlador en Visual Studio 2017

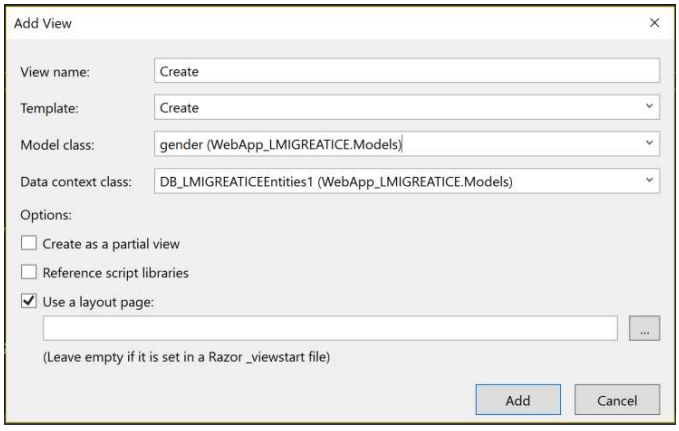


Figura 1.3 Formulario para añadir una Vista en Visual Studio 2017

Generalmente, la validación de datos en el lado del cliente siempre ha sido un reto para los desarrolladores, puesto que no se trata únicamente de implementar la lógica de validación, sino también de mostrar los mensajes adecuados a los usuarios para proveer una retroalimentación que les permita saber si los datos que ingresan son o no válidos.

Data Annotations permite la descripción de las reglas que se desean aplicar a las propiedades del Modelo. ASP.NET MVC se encarga de hacer cumplir estas reglas y mostrar los mensajes apropiados a los usuarios [10].

Para utilizar esta herramienta, es necesario agregar un espacio de nombres en el código de la aplicación a través de la directiva `using`, de la siguiente manera:

```
using System.ComponentModel.DataAnnotations;
```

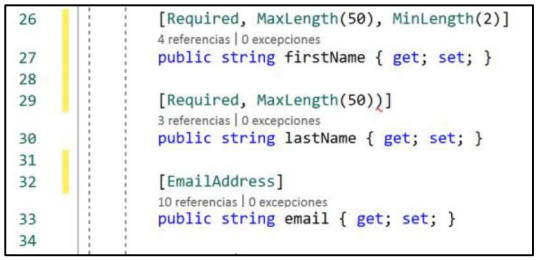
En la Tabla 1.1 se muestran los *Data Annotations* más utilizados:

Tabla 1.1 *Data Annotations* comunes

Data Annotation	Función
`Required`	El campo indicado debe llenarse obligatoriamente.
`Range`	Define un valor máximo y mínimo de valores numéricos.
`DisplayName`	Indica el texto que se va a mostrar para el campo indicado.
`MinLength`	Indica la longitud mínima que debe tener una cadena (`string`).
`MaxLength`	Indica la longitud máxima que debe tener una cadena (`string`).
`EmailAddress`	Comprueba que la cadena ingresada tenga un formato válido de correo electrónico.

En el Código 1.3 se muestra un ejemplo de la utilización de *Data Annotations.* Se puede observar que los *Data Annotations* son colocados encima de los atributos de la clase para validar los datos del Modelo.

En este caso, para el atributo `firstName` se indica que es un campo requerido cuya longitud máxima debe ser de 50 caracteres y la longitud mínima debe ser de 2 caracteres (línea 26). Para el atributo `lastName` se especifica que es un campo requerido cuya longitud máxima debe ser de 50 caracteres (línea 29). Finalmente, en el atributo `email` se utiliza `EmailAddress` para comprobar que el formato de la cadena de texto ingresada sea congruente con el formato de un correo electrónico (línea 32).



Código 1.3 Utilización de *Data Annotations* en los atributos de una clase

1.3.2.2 Vista

Es la encargada de presentar la información al cliente (usuario). La Vista se encarga de todo lo que tiene que ver con la interfaz gráfica de usuario (GUI), cómo se muestran los datos y cómo se maneja la interacción de la aplicación con el usuario.

Las Vistas contienen código HTML y CSS.

También, se dispone de algunas herramientas que facilitan la codificación en las Vistas, como las que se detallan a continuación:

RAZOR: Es una sintaxis que permite embeber código basado en el servidor (C# o VB.Net) dentro de páginas web [11].

ASP.NET MVC implementa RAZOR para combinar código desarrollado en C# o VB.Net con HTML dentro de las Vistas para generar páginas web dinámicas.

RAZOR no es un lenguaje de programación. Es una lenguaje de marcado en el lado del servidor [12].

La sintaxis RAZOR tiene las siguientes características:

- Es compacta, lo que permite minimizar el número de caracteres requeridos para escribir código.
- Admite Intellisense[11] para el autocompletado de código en Visual Studio.
- Las Vistas que mezclan código C# con HTML tienen la extensión de archivo `.cshtml`. Mientras que las Vistas que combinan VB.Net con HTML tienen la extensión `.vbhtml`.

[11] Intellisense: Es una herramienta de Visual Studio que permite el autocompletado de código, información de parámetros, información rápida y listas de miembros.

En el Código 1.4 se muestra un ejemplo simple de Vista en ASP.NET MVC que utiliza RAZOR para combinar código C# con HTML.

```
59  @{
60      string var = "";
61      string title = "RAZOR";
62      if (title != null)
63      {
64          var = title + "Example";
65      }
66
67  }
68
69  <p> Message: @var  </p>
```

Código 1.4 Sintaxis RAZOR en una Vista

Como se puede observar, los bloques de código C# inician con el símbolo `@`, que indica el inicio de la sintaxis RAZOR. Posteriormente se tienen los símbolos `{}`, dentro de los cuales se escribe el código, que puede ser una sentencia única o un bloque completo de código.

Cada sentencia de código dentro de `{}` debe finalizar con punto y coma.

Para acceder al valor de una variable desde HTML, es necesario utilizar el símbolo `@` seguido del nombre de la variable.

RAZOR es lo suficientemente inteligente como para intercambiar entre el código C# y HTML o viceversa.

La sintaxis RAZOR es utilizada en las Vistas para generar elementos HTML a través de la clase `HtmlHelper`. Esta clase contiene diferentes métodos, conocidos como *HTML Helpers*, que permiten generar controles HTML mediante código.

`HtmlHelper` enlaza un objeto del Modelo a elementos HTML para mostrar el valor de las propiedades del Modelo en dichos elementos HTML, y también asigna el valor contenido en los elementos HTML a las propiedades del Modelo al enviar un formulario web.

El valor retornado por los *HTML Helpers* es un `string`.

La clase `HtmlHelper` no debe ser usada en el Modelo ni en los Controladores, puesto que está diseñada para generar elementos correspondientes a la interfaz de usuario [13]. En la Tabla 1.2 se describen algunos *HTML Helpers* estándar:

Tabla 1.2 *HTML Helpers* estándar

Método	Función
`Html.Label`	Genera una etiqueta de texto estático
`Html.TextBox`	Genera un cuadro de texto
`Html.Editor`	Similar a `HTML.TextBox`, pero el elemento generado varía su presentación de acuerdo al tipo de dato que maneja.
`Html.Password`	Genera un cuadro de texto para contraseñas, en el que su contenido no se muestra al usuario.
`Html.ActionLink`	Genera un hipervínculo que se enlaza a una acción específica en el Controlador.
`Html.DropDownList`	Genera una lista opciones que se despliegan para su selección.
`Html.CheckBox`	Genera un control mediante el cual se pueden seleccionar varias opciones de un conjunto predefinido, a través de una casilla de verificación.
`Html.RadioButton`	Genera un control que permite seleccionar una única opción entre un conjunto de opciones predefinidas.
`Html.Display`	Muestra texto HTML.

Para entender la utilización de los *HTML Helpers*, se presenta un ejemplo en el Código 1.5, donde se muestra una Vista en la que el usuario debe ingresar manualmente el año, para que el sistema realice una búsqueda de cierta información respecto al año ingresado. Inicialmente se muestra la etiqueta con texto estático (línea 72). Luego, se genera un cuadro de texto para que el usuario digite el año de búsqueda (línea 74). También, se genera un hipervínculo que permite retornar a la página inicial y realizar una nueva búsqueda (línea 81).

Bootstrap: Es el *framework front-end* más popular para el desarrollo de sitios web de tipo *responsive*[12].

Bootstrap incluye plantillas de diseño basadas en HTML y CSS para tipografía, formularios, botones, tablas y muchos otros, así como complementos opcionales de JavaScript [14].

[12] *Responsive*: Un diseño web *responsive* es aquel que se ajusta automáticamente para verse bien en todos los dispositivos, desde teléfonos móviles hasta computadores de escritorio.

Bootstrap facilita el trabajo de los programadores al eliminar la necesidad de escribir grandes cantidades de código CSS, lo cual puede ser bastante tedioso. En cambio, les permite enfocarse en el desarrollo del sitio web, ahorrándoles tiempo en la personalización de la interfaz gráfica.

```
<div class="form-group" id="yearGroup">
 @Html.Label("Year", "Year", htmlAttributes: new { @class = "control-label col-md-2" })
   <div class="col-md-10">
     @Html.DropDownListFor(model => model.year, ViewBag.years as SelectList, htmlAttributes: new { @class = "form-control" })
   </div>
 </div>
</div>
<div class="form-group">
 <div class="col-md-offset-2 col-md-10">
   <input type="submit" value="Search" class="btn btn-default wine-button" />
      @Html.ActionLink("Clear", "Search", null, new { @class = "btn btn-default wine-button" })
   </div>
  </div>
```

Código 1.5 Ejemplo de utilización de *HTML Helpers* en una Vista

jQuery: Es una librería de JavaScript cuyo objetivo es simplificar el uso de este lenguaje de programación en el desarrollo web. jQuery toma una gran cantidad de tareas comunes que requerirían muchas líneas de código JavaScript y las agrupa en métodos a los que se puede llamar mediante una sola línea de código [15].

jQuery permite añadir fácilmente un comportamiento dinámico a sitios web estáticos. El valor de jQuery radica en la simplicidad para agregar interactividad a un sitio web, como por ejemplo la transición entre páginas, un menú de navegación que se adapta al tamaño de la pantalla, un *slider*[13], animaciones, entre muchos otros.

1.3.2.3 Controlador

Funciona como una conexión entre el Modelo y la Vista. El Controlador recibe peticiones e interpreta las acciones del usuario y actúa en concordancia con lo solicitado. Es el encargado de gestionar las acciones e interactuar con los otros dos componentes.

El Controlador permite responder a las solicitudes HTTP del usuario y retornar información al navegador. Es el encargado de manejar la lógica de la aplicación web, interactuar con el Modelo y seleccionar la Vista adecuada [8].

[13] *Slider*: En desarrollo web, un slider permite la transición y animación de imágenes.

Hay dos tipos de Controladores, los síncronos y los asíncronos:

Controladores Síncronos: Tradicionalmente utilizados en aplicaciones web, pues la aplicación sigue un flujo síncrono. Es decir, se hace una petición al servidor y se espera por una respuesta, sea cual sea el tiempo que tarde en recibirse. Hasta que no se termine de procesar la primera solicitud, no se pueden atender otras solicitudes [16].

Controladores Asíncronos: Permiten implementar métodos asíncronos, gracias a lo cual se consigue liberar el hilo de procesamiento para que otros procesos puedan utilizarlo en caso de ser necesario, evitando un bloqueo. Este tipo de controlador es muy útil cuando se desea que varios procesos se ejecuten de manera simultánea, puesto que, si no se libera el hilo y se está ejecutando un proceso muy pesado, puede ocurrir lo que se conoce como inanición de procesos.

Los Controladores tienen métodos con diferentes tipos de acción. Cada tipo de acción retorna resultados distintos. La clase base es `ActionResult` y representa el resultado de una acción en el Controlador [9].

En la Tabla 1.3 se muestran los distintos tipos de acción en ASP.NET MVC.

Tabla 1.3 Tipos de Acción en ASP.NET MVC

Tipo de Acción	Método	Descripción
`ViewResult`	`View()`	Retorna una Vista.
`PartialViewResult`	`PartialView()`	Retorna una Vista parcial.
`JsonResult`	`Json()`	Retorna datos en formato JSON.
`RedirectResult`	`Redirect()`	Redirige hacia una nueva URL[14] especificada.
`ContentResult`	`Content()`	Retorna texto simple.
`RedirectToRouteResult`	`RedirectToAction()`	Redirecciona a otra ruta/acción en el mismo Controlador o en otro.
`FileResult`	`File()`	Retorna contenido de archivos binarios.
`EmptyResult`	–	No retorna nada

[14] URL (*Uniform Resource Locator*): Es la dirección de la ubicación de un recurso en Internet.

Para enviar los datos entre los Controladores y las Vistas, o entre distintas acciones en uno o varios Controladores, se utilizan contenedores de datos como los que se detallan a continuación:

- **`ViewBag`**: Es un contenedor que permite enviar una pequeña cantidad de datos temporales que no están incluidos en el Modelo desde el Controlador hacia la Vista [17].

  Un ejemplo de utilización de `ViewBag` se muestra en la Figura 1.4, donde se observa que en la Vista se puede acceder al valor enviado desde el Controlador utilizando el símbolo `@,` que denota la utilización de sintaxis RAZOR, explicada anteriormente.

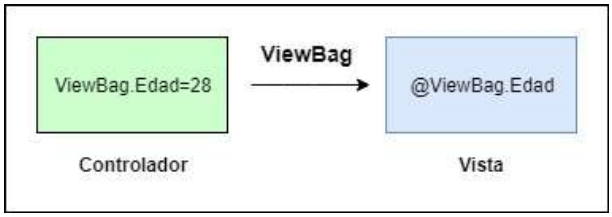


Figura 1.4 Ejemplo de utilización de `ViewBag`

  A `ViewBag` se puede asignar cualquier número de propiedades y valores. De igual manera, se le puede asignar un objeto de tipo primitivo o complejo.

  Es importante recalcar que `ViewBag` solo transfiere datos desde el Controlador hacia la Vista, y no viceversa. Entonces, si ocurre una redirección, los valores del `ViewBag` serán nulos, puesto que los valores del `ViewBag` únicamente se mantienen durante la solicitud HTTP actual.

- **`ViewData`**: Es un contenedor al igual que `ViewBag`. `ViewData` es un diccionario que puede contener pares clave-valor, donde las claves deben ser de tipo `string` [18]. En la Figura 1.5 se muestra un ejemplo de utilización de `ViewData`.

  Al igual que `ViewBag`, `ViewData` solo transfiere datos desde el Controlador hacia la Vista y no viceversa. Únicamente es válido durante la solicitud HTTP actual.

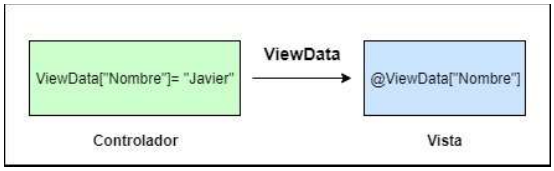


Figura 1.5 Ejemplo de utilización de `ViewData`

Al utilizar `ViewData` es necesario realizar una conversión en la Vista del tipo de dato contenido en el `ViewData` antes de utilizarlo, esto se debe a que `ViewData` siempre devuelve un objeto.

- **`TempData`:** Se utiliza para almacenar datos temporales que se pueden usar en la solicitud HTTP posterior. Los datos se borrarán luego de la finalización de la siguiente solicitud [19].

  Es un tipo de diccionario muy útil en redirecciones o cuando se quieren transferir datos desde un método de acción hasta otro método de acción, ya sea en el mismo controlador o en otro. También se puede usar para enviar información desde el Controlador a la Vista.

  En la Figura 1.6 se muestra un ejemplo de utilización de `TempData`, como se observa, en la segunda solicitud HTTP se puede acceder a los datos almacenados en un `TempData` de la primera solicitud. Sin embargo, estos datos ya no pueden ser accedidos en una tercera solicitud debido a que el contenedor se vacía luego de la segunda solicitud.

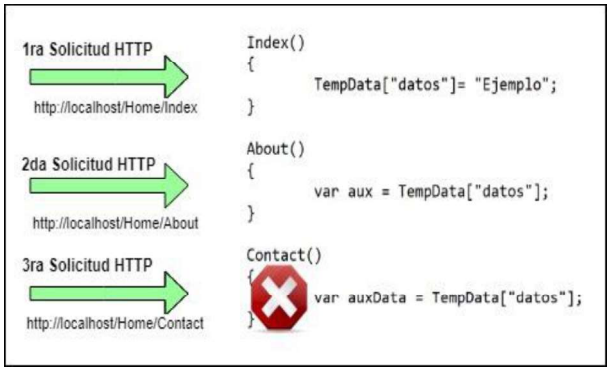


Figura 1.6 Ejemplo de utilización de `TempData`

Es necesario realizar una conversión del tipo de dato antes de utilizar el valor almacenado en un `TempData`.

Para acceder a los datos almacenados en un `TempData` en una tercera solicitud, se puede utilizar `TempData.Keep()`

- **`Session`:** Usada para almacenar datos que se pueden enviar entre distintos Controladores, así como también para enviar datos desde un Controlador hacia una Vista [20].

  A diferencia de `TempData`, los datos contenidos en una `Session` nunca expiran.

  Esto se puede observar en la Figura 1.7, donde se muestra que sin importar cuántas solicitudes HTTP se hagan, los valores contenidos en una `Session` se mantienen.

  Al igual que en `TempData`, el valor contenido en una `Session` debe ser convertido al tipo de dato necesario antes de procesarlo.

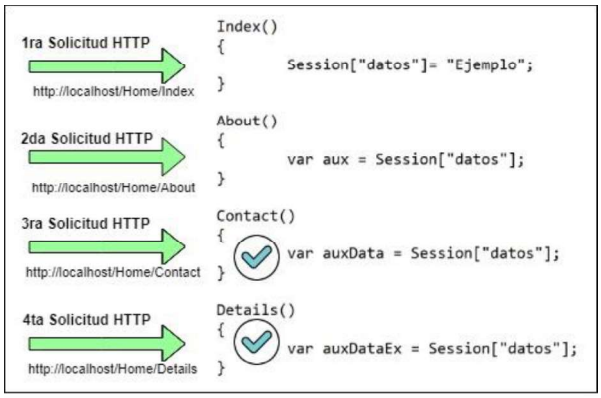


Figura 1.7 Ejemplo de utilización de `Session`

Para desarrollar una aplicación web utilizando ASP.NET es necesario contar con un IDE[15], un servidor IIS[16] y el *framework* .NET.

[15] IDE (*Integrated Development Environment*): Es una aplicación informática que empaqueta dentro de sí un editor de código, un compilador, un depurador, un constructor de interfaz gráfica, entre otros. El IDE facilita al programador el desarrollo de aplicaciones.
[16] IIS (*Internet Information Services*): Es un servidor web y un conjunto de servicios para el sistema operativo Windows.

- IDE (*Integrated Development Environment*): Visual Studio 2017 es un IDE elaborado por Microsoft que soporta el desarrollo de aplicaciones con varios lenguajes de programación como C#, C++, Visual Basic, entre otros.

- IIS (*Internet Information Servicies*): Es un servidor web de Microsoft incorporado en los sistemas operativos Windows, que permite albergar sitios web.

  IIS acepta solicitudes de clientes remotos y retorna la respuesta apropiada. Esto le permite compartir y desplegar información ya sea como sitios web estáticos codificados en HTML, o dinámicos como aplicaciones en ASP.NET o páginas PHP.

  IIS se ayuda de varios protocolos para su trabajo, como HTTP (*Hypertext Transfer Protocol*) para la comunicación entre usuarios y el servidor; HTTPS (*Hypertext Transfer Protocol Secure*), una variante de HTTP que implementa cifrado en la comunicación para añadir seguridad; FTP (*File Transfer Protocol*) que permite la transferencia de archivos; SMPT (*Simple Mail Transfer Protocol*) para enviar y recibir correos electrónicos [21]. El servidor web IIS implementa el modelo de generar un nuevo hilo para cada solicitud que le envían. Es decir, cada solicitud será procesada en un hilo diferente, y se ejecutará como un proceso separado [22].

- .NET *Framework*: Es un marco de trabajo elaborado por Microsoft para el desarrollo de aplicaciones. Los principales componentes de este *framework* son: el conjunto de lenguajes de programación, la biblioteca de clases y CLR (*Common Language Runtime*)[17]. Este último se encarga de compilar el código fuente de cualquier lenguaje de programación soportado por .NET a un código intermedio, para posteriormente transformarlo en código máquina y ejecutarlo, de esta manera se consigue independencia e interoperabilidad entre los lenguajes de programación.

Al crearse un nuevo proyecto en Visual Studio para el desarrollo de una aplicación web ASP.NET MVC, se generan los archivos y carpetas necesarios para que la aplicación funcione.
Un resumen con los principales archivos de configuración se presenta en la Tabla 1.4 [23].
La estructura de directorios por defecto se muestra en la Tabla 1.5 [16].

[17] CLR (*Common Language Runtime*): Es un entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes de programación soportados por .NET.

Tabla 1.4 Archivos de configuración de un proyecto ASP.NET MVC

`Startup.cs`	Contiene una clase que se ejecuta al iniciar la aplicación web y también cada vez que se realiza una petición/respuesta HTTP.
`Global.asax`	Es un archivo que contiene controladores de eventos para manejar eventos a nivel de aplicación como: inicio de aplicación, inicio y fin de sesión, error en la aplicación, entre otros.
`Web.config`	Es un archivo que contiene todas las configuraciones de un sitio web almacenadas en XML y separadas del código de la aplicación.
`packages.config`	Es manejando por NuGet[18] para mantener un seguimiento de qué paquetes y versiones están instalados en la aplicación web.

Tabla 1.5 Directorios por defecto de un proyecto ASP.NET MVC

`App_Data`	Contiene archivos de datos, archivos XML y archivos de bases de datos.
`App_Start`	Contiene archivos que se ejecutan para realizar todas las configuraciones necesarias al iniciarse la aplicación.
`Content`	Contiene archivos CSS, imágenes e íconos.
`Controllers`	Contiene archivos para los Controladores del proyecto.
`fonts`	Contiene archivos de fuentes personalizadas para la aplicación.
`Models`	Contiene archivos de clases del Modelo. Generalmente las clases del Modelo contienen atributos públicos.
`Scripts`	Contiene archivos JavaScript para la aplicación. ASP.NET MVC incluye archivos JavaScript para Bootstrap y jQuery.
`Views`	Contiene archivos HTML para las Vistas del Proyecto. Cada Controlador tiene una subcarpeta dentro de `Views` para sus respectivas Vistas.

[18] NuGet: Es un gestor de paquetes de Microsoft que permite albergar, compartir y consumir paquetes de código reutilizable en la plataforma .NET.

1.3.3 BASES DE DATOS RELACIONALES

Una base de datos es una colección de información organizada de tal manera que puede ser fácilmente accedida, administrada y actualizada [24].

En el diseño de la base de datos es muy importante la abstracción de la información, destacando las características más importantes de la misma y llevando un problema del mundo real a esquemas, este proceso se conoce como modelamiento de datos.

Entonces, podría decirse que un modelo de datos es una representación gráfica de estructuras de datos del mundo real que permiten un mejor entendimiento de la información a manejar.

Una base de datos relacional es aquella que cumple con el modelo de datos relacional.

El modelo relacional representa una base de datos como una colección de relaciones. De manera simplificada, una relación, también llamada tabla, puede ser definida como una estructura bidimensional compuesta de la intersección de filas y columnas. Cada fila dentro de la tabla se denomina tupla, y cada columna de la tabla representa un atributo. En otras palabras, el modelo relacional hace uso de un grupo de tablas para representar los datos, donde el nombre de la tabla representa la entidad, las columnas de la tabla corresponden a atributos de dicha entidad y las tuplas o filas contienen información acerca de una entidad particular. Todos los valores de una columna tienen el mismo tipo de dato.

Cada modelo de datos consta de los siguientes componentes [25]:

- **Entidad**: representa un tipo particular de objeto del mundo real. Cada entidad es única y distinta.
- **Atributo**: representa una característica de una entidad.
- **Relación**: describe una asociación entre entidades. Hay 3 tipos de relaciones entre entidades:

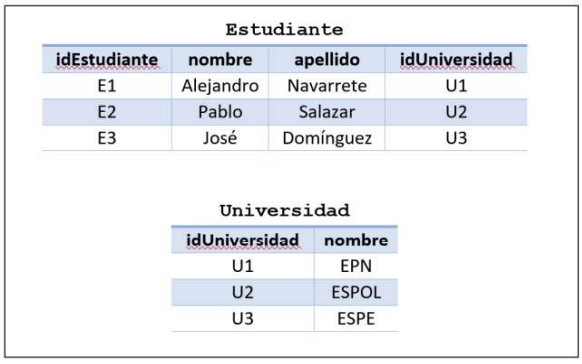
  Uno a uno. – Una instancia de la entidad está asociada con solo una instancia de la entidad relacionada.

  Uno a varios. – Una instancia de la entidad está asociada con varias instancias de la entidad relacionada.

  Varios a varios. - Una instancia de la entidad está asociada con muchas instancias de la entidad relacionada y, una instancia de la entidad relacionada está asociada con muchas instancias de la primera entidad.

- **Restricción**: representa una regla o limitación que se coloca a los datos. Las restricciones ayudan a garantizar la integridad de los datos.

En el ejemplo de la Figura 1.8, en la tabla `Estudiante`, la columna `idEstudiante` representa lo que se conoce como clave primaria, que no es más que un identificador único para cada una de las tuplas de dicha tabla [25]. Mientras que, la columna `idUniversidad` representa a una clave foránea, que es una referencia a la clave principal de la tabla `Universidad`. La clave foránea permite relacionar dos tablas, es decir, a través de la clave foránea, una tabla secundaria puede acceder a los atributos de una tabla primaria [25]. En este caso, a través de la clave foránea, la tabla `Estudiante` puede acceder a atributos de la tabla `Universidad`.



Estudiante

idEstudiante	nombre	apellido	idUniversidad
E1	Alejandro	Navarrete	U1
E2	Pablo	Salazar	U2
E3	José	Domínguez	U3

Universidad

idUniversidad	nombre
U1	EPN
U2	ESPOL
U3	ESPE

Figura 1.8 Ejemplo de modelo relacional

Los diagramas relacionales permiten visualizar de forma esquematizada el diseño de una base de datos relacional. En la Figura 1.9 se observa el diagrama relacional para el ejemplo anterior. En ella se detallan las entidades `Universidad` y `Estudiante`, sus respectivos atributos y la relación existente entre las entidades, en este caso, uno a varios.

Para la gestión de una base de datos relacional se utiliza SQL[19], que está compuesto de comandos que permiten a los usuarios crear bases de datos relacionales, tablas, realizar diversos tipos de manipulación y administración de datos, y consultar la base de datos para extraer información útil [26].

[19] SQL (*Structured Query Language*): es un lenguaje estándar para el almacenamiento y manipulación de datos en bases de datos relacionales.

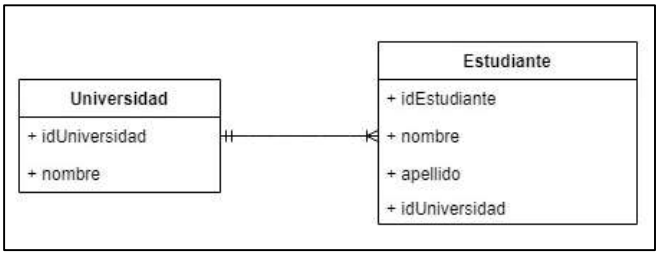


Figura 1.9 Ejemplo de diagrama relacional

SQL permite especificar lo que debe hacerse sin especificar el cómo debe hacerse, lo que facilita al usuario la creación y manejo de las bases de datos.

En el Código 1.6 se muestra un ejemplo de comandos SQL.

Inicialmente se indica que se utilizará la base de datos `DB_LMIGREATICE`; la siguiente línea indica la creación de una tabla denominada `dataType`, la misma que contiene los atributos `idDataType`, `nameDataType` y `stateR`. También se indican los tipos de datos y las restricciones que tiene cada atributo.

```
use DB_LMIGREATICE

create table dataType(
    idDataType int identity(1,1) primary key,
    nameDataType varchar(15) not null,
    stateR bit not null default 1
);
```

Código 1.6 Ejemplo de comandos SQL

1.3.4 ENTITY FRAMEWORK (EF)

Es una tecnología de acceso a datos de Microsoft que forma parte del *framework* .NET. En las primeras versiones de .NET, que no implementaban Entity Framework, había una necesidad de mezclar el código del programa, elaborado por ejemplo en C#, con sentencias SQL para realizar el acceso a los datos albergados en una base de datos relacional. Esto a su vez requería que los desarrolladores tengan un conocimiento sólido del lenguaje de programación y de SQL, además, se mezclaban conceptos que nada tenían

que ver entre sí, impidiendo que se aprovechen de manera eficiente los beneficios que otorga la programación orientada a objetos.

Debido a estas y otras dificultades, Microsoft desarrolló para su plataforma .NET, un marco de trabajo que contiene bibliotecas especializadas para el acceso a datos, denominado Entity Framework. Entity Framework es un ORM[20], por lo cual permite generar clases a partir de tablas de una base de datos y sus relaciones, o viceversa, crear las entidades necesarias en una base de datos partiendo de una jerarquía de clases. Todo esto se hace de manera transparente al usuario [27].

Entity Framework otorga una serie de beneficios que facilitan el desarrollo de aplicaciones que interactúan con bases de datos:

- No es necesario que se escriba código SQL, debido a que se puede utilizar consultas basadas en LINQ[21] para retornar información de la base de datos.
- Permite aprovechar de mejor manera las ventajas de la programación orientada a objetos.
- Fomenta la reutilización de código y su mantenimiento, puesto que se tiene un único modelo que es utilizable en toda la aplicación. No se mezclan consultas SQL con código.

Existen tres modos o enfoques de trabajo en Entity Framework: *Code First*, *Model First* y *Database First,* los cuales se explican a continuación [28] y [29]:

- *Code First*. – En este enfoque se parte de clases ya definidas en el código de la aplicación para que EF genere la base de datos, tablas y demás para encajar las clases en ellas.
- *Model First*. – Se crea el modelo de datos visualmente, con las entidades, relaciones y atributos, y EF se encarga de generar la base de datos y también las clases en el código de la aplicación.
- *Database First*. – Este enfoque considera que ya está creada la base de datos y, a partir de la misma, EF genera las clases necesarias en la aplicación. Estrictamente, EF asocia las tablas de la base de datos con clases de la aplicación y las columnas de las tablas son asociadas con atributos dentro de las clases.

[20] ORM (*Object-Relational Mapper*): Herramienta especializada en el acceso a datos que permite asociar las estructuras de una base de datos relacional con estructuras lógicas de la programación orientada a objetos.
[21] LINQ (*Language Integrated Query*): Conjunto de herramientas de Microsoft que permiten realizar todo tipo de consultas a distintas fuentes de datos.

En la Figura 1.10 se muestra en dónde opera Entity Framework dentro de la aplicación, entre la base de datos y las clases de dominio (entidades de negocio).

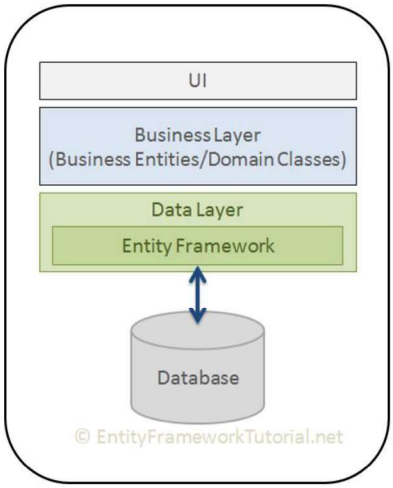


Figura 1.10 Operación de Entity Framework [30]

En *Database First*, Entity Framework crea primeramente un Modelo de Datos de la Entidad (EDM[22]) para una base de datos existente.

El EDM es usado para todas las operaciones relacionadas con la base de datos.

En la Figura 1.11 se muestra un ejemplo de EDM creado por Entity Framework en Visual Studio 2017 a partir de una base de datos ya existente. Este modelo se almacena en un archivo con extensión `.edmx`.

El EDM consta a su vez de tres partes principales:

- Modelo conceptual (*Conceptual Model*): Es una representación de la estructura de los datos como entidades y relaciones. [31].
- Modelo de almacenamiento (*Storage Model*): Contiene el diseño de la base de datos que incluye tablas, vistas, procedimientos almacenados, claves, etc.
- Asociación (*Mapping*): Contiene información acerca de cómo el modelo conceptual se asocia al modelo de almacenamiento.

[22] EDM (*Entity Data Model*): Es un modelo de datos que describe las entidades y las relaciones existentes entre ellas.

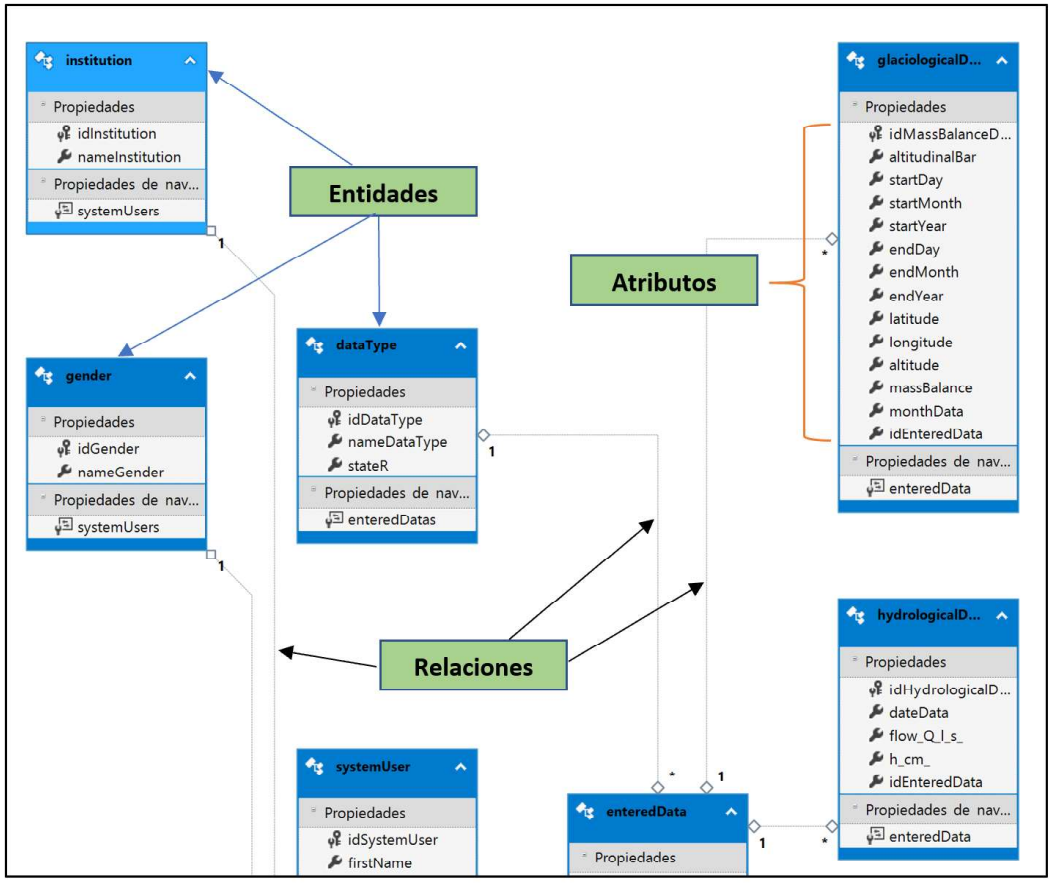


Figura 1.11 Ejemplo de EDM creado por Entity Framework

En el caso de las aplicaciones web, el EDM añade al archivo `Web.config` la cadena de conexión, tal como se muestra en el Código 1.7.

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;
       AttachDbFilename=|DataDirectory|\aspnet-WebApp_LMIGREATICE-20180823024639.mdf;
       Initial Catalog=aspnet-WebApp_LMIGREATICE-20180823024639;
       Integrated Security=True" providerName="System.Data.SqlClient" />
  <add name="DB_LMIGREATICEEntities1" connectionString="metadata=res://*/
       Models.LmiGreatIceModel.csdl|res://*/Models.LmiGreatIceModel.ssdl|res:
       //*/Models.LmiGreatIceModel.msl;provider=System.Data.SqlClient;
       provider connection string=&quot;data source=JAAN\SQLSERVER2017;initial catalog=DB_LMIGREATICE;
       integrated security=True;multipleactiveresultsets=True;application name=EntityFramework&quot;"
       providerName="System.Data.EntityClient" />
</connectionStrings>
```

Código 1.7 Parte del archivo `Web.config` de la aplicación donde se muestra la cadena de conexión a la base de datos

Al generarse el EDM, en el archivo `.edmx` se crea una sola Clase Contexto (*Context Class*) y una Clase Entidad (*Entity Class*) para cada una de las tablas de la base de datos.

La Clase Contexto es muy importante y representa una sesión con la base de datos subyacente. Se deriva de `DbContext`[23] (Código 1.8, línea 18) y se utiliza para consultar o guardar datos en la base de datos, así como para configurar clases de dominio, seguimiento de cambios, entre otros.

La Clase Contexto se almacena dentro de un archivo con extensión `Context.tt` que a su vez está dentro de un archivo con extensión `.edmx`.

```
18  public partial class DB_LMIGREATICEEntities1 : DbContext
19  {
        8 referencias | 0 excepciones
20      public DB_LMIGREATICEEntities1()
21          : base("name=DB_LMIGREATICEEntities1")
22      {
23      }
24
        0 referencias | 0 excepciones
25      protected override void OnModelCreating(DbModelBuilder modelBuilder)
26      {
27          throw new UnintentionalCodeFirstException();
28      }
29
        2 referencias | 0 excepciones
30      public virtual DbSet<country> countries { get; set; }
        3 referencias | 0 excepciones
31      public virtual DbSet<dataType> dataTypes { get; set; }
```

Código 1.8 Ejemplo de Clase Contexto

El archivo con extensión `Context.tt` es una plantilla T4[24] que genera una Clase Contexto cada vez que se cambia el modelo de datos de la entidad (EDM). El nombre por defecto de la Clase Contexto está dado por el nombre de la base de datos seguido de la palabra *Entities*, tal como se muestra en la Figura 1.12, donde `DB_LMIGREATICE` es el nombre de la base de datos.

Las Clases Entidad representan a cada una de las tablas de la base de datos. Se encuentran dentro de un archivo con extensión `.tt`, el cual a su vez está dentro de un archivo con extensión `.edmx`.

---

[23] `DbContext`: Es una clase dentro del espacio de nombres `System.Data.Entity` que representa un puente entre las Clases Entidad y la base de datos. Permite interactuar a la aplicación con la base de datos.

[24] Plantilla T4: Es una mezcla de bloques de texto y lógica de control que puede generar un archivo de texto. La lógica de control se escribe como fragmentos de código en C# o Visual Basic [49].

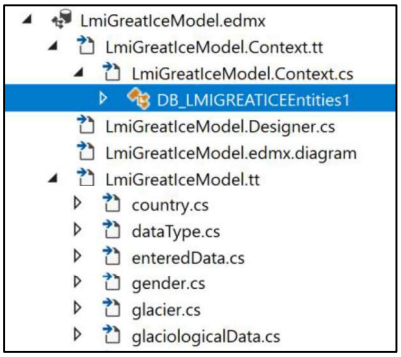


Figura 1.12 Localización de Clase Contexto y Clases Entidad

En la Figura 1.12 se muestran las Clases Entidad generadas por Entity Framework y que representan a las tablas de la base de datos: `country.cs`, `dataType.cs`, `enteredData.cs`, `gender.cs`, `glacier.cs` y `glaciologicalData.cs`.

Una vez que el modelo de datos de la entidad (EDM) ha sido creado a partir de la base de datos, el trabajo de Entity Framework se puede resumir de la siguiente manera:

- Para insertar datos en la base, Entity Framework utiliza la sentencia: `Context.Add (objeto)`
- Para eliminar datos, se utiliza la sentencia: `Context.Remove(objeto)`
- Para actualizar datos, se utiliza la sentencia: `Context.Update (objeto)`

Luego de cualquiera de las sentencias anteriores, es necesario llamar al método `Context.SaveChanges()`, para guardar los cambios realizados en la base de datos.

Como se mencionó anteriormente, Entity Framework elimina la necesidad de que, en el código de programación, las consultas o sentencias que tengan que ver con la base de datos se definan usando SQL.

Para búsquedas que requieran retornar información de la base de datos, se puede utilizar expresiones lambda[25] o consultas del tipo LINQ.

[25] Expresiones lambda: Son funciones anónimas que se usan frecuentemente para crear delegados en LINQ. No tienen modificadores de acceso y devuelven el resultado de la evaluación de la condición que se indica.

1.3.4.1 LINQ

Es un conjunto de herramientas que proveen una sintaxis de consulta uniforme para recuperar datos de diferentes fuentes y formatos [32].

La sintaxis de consulta de LINQ permite realizar operaciones de filtrado, ordenamiento y agrupación de datos con un mínimo de código.

Además, se puede utilizar los mismos patrones de consulta para aplicarlos en distintas fuentes de datos como colecciones, arreglos, documentos XML, bases de datos SQL, entre otros.

En el Código 1.9 se muestra un ejemplo de consulta LINQ en C# que utiliza Entity Framework para acceder a la base de datos, específicamente a la tabla `enteredDatas`, y retorna como resultado la/s tupla/s que tienen el valor de 1 en la columna `idDataType` de dicha tabla.

```
479
480         var lista = from doc in db.enteredDatas
481                     where doc.idDataType == 1
482                     select doc;
```

Código 1.9 Ejemplo de consulta LINQ

También se puede acceder a los datos mediante la utilización de expresiones lambda, que utilizan una sintaxis más directa para interactuar con LINQ, como se muestra en el Código 1.10.

En este caso, la expresión lambda busca en la tabla `enteredDatas` el registro cuyo atributo `idDataType` sea igual a 1, lo recoge y lo asigna a la variable `doc`.

```
480         var doc = db.enteredDatas.Where(a => a.idDataType == 1).FirstOrDefault();
```

Código 1.10 Ejemplo de expresión lambda

El flujo de trabajo de Entity Framework se muestra en la Figura 1.13. En el primer caso, se considera la ejecución de consultas para el retorno de información desde la base de datos.

En el segundo caso, se plantea la ejecución de sentencias de inserción, eliminación y modificación de datos en la base de datos.

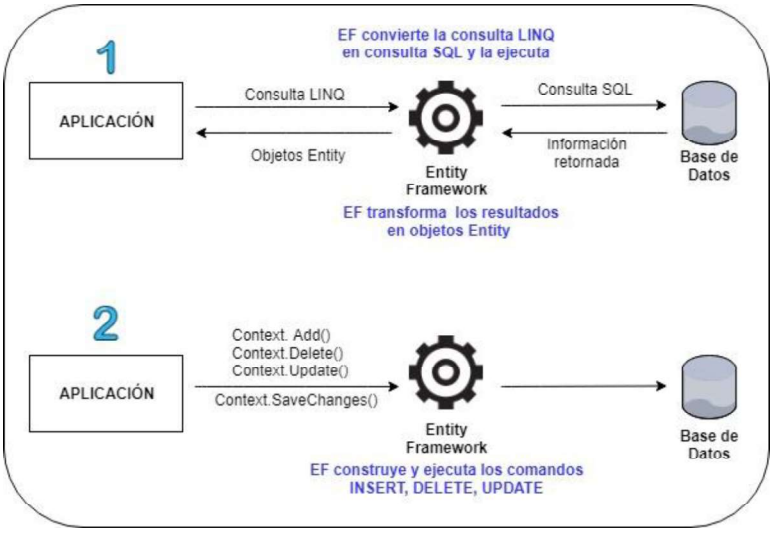


Figura 1.13 Flujo de trabajo de Entity Framework

1.3.5 SERIES TEMPORALES

Una serie temporal puede definirse como una colección de observaciones de una variable recogidas secuencialmente en el tiempo. Estas observaciones se suelen recoger en instantes de tiempo equiespaciados [33].

El estudio de series temporales permite analizar el cambio de la variable medida en un intervalo de tiempo, es decir, la serie temporal describe la evolución histórica de una variable o magnitud [34].

Además, gracias a las series temporales se puede predecir valores futuros que tomará dicha variable.

Ejemplos de uso de series temporales pueden ser: lluvia recogida en un sitio específico diariamente, nivel de radiación a lo largo de un día en una localidad, variaciones de caudal en un río, entre muchos otros.

Frecuentemente, las series temporales son expresadas en gráficas temporales, donde el valor de la serie se ubica en el eje de ordenadas (eje Y), y los tiempos en el eje de abscisas (eje X).

En la Figura 1.14 se muestra un ejemplo de gráfica temporal, que contiene datos de Temperatura (ºC) a distintas horas del día, tomadas desde el 08/01/2019 al 09/01/2019 en Santiago del Estero (Argentina). El eje de las abscisas muestra los tiempos en los que fueron tomadas las medidas, en este caso, la Temperatura fue tomada a lo largo de 24 horas, con una muestra cada hora, desde las 18:00 horas del 08/01/2019 hasta las 18:00 horas del 09/01/2019. El eje de las ordenadas muestra por su parte los valores de las temperaturas tomadas en cada hora. Al unir los puntos resulta una curva como la mostrada.

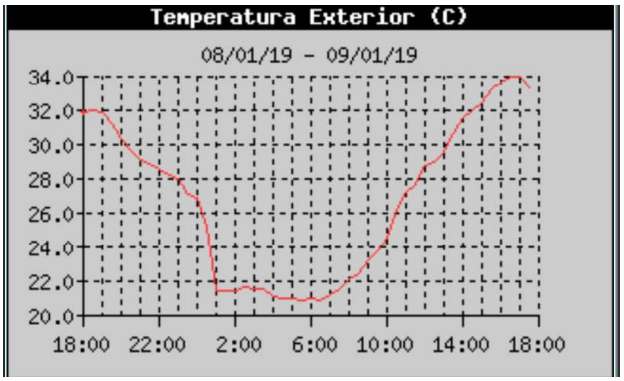


Figura 1.14 Ejemplo de gráfica temporal: Temperatura vs tiempo [35]

1.3.6 MAPAS INTERACTIVOS CON LEAFLET Y GEOJSON

Leaflet es una librería de código abierto utilizada en la implementación de mapas interactivos.

Funciona eficientemente en todas las principales plataformas de escritorio y móviles [36]. Surgió como una alternativa a otras alternativas similares pero que tienen un costo una vez que se supera cierto umbral de utilización, como MapBox[26] y Google Maps[27].

Leaflet es una librería gratuita de JavaScript que utiliza los mapas disponibles en OpenStreetMap[28] [37]. Es bastante flexible y permite agregar capas personalizadas, marcadores, controles de estilo e integración de eventos [38].

[26] MapBox: Es una librería que permite desarrollar mapas en línea personalizados para sitios web y aplicaciones.
[27] Google Maps: Es un servicio para mapas web desarrollado por Google. Ofrece imágenes satelitales, mapas de calles, vistas panorámicas, condiciones de tráfico en tiempo real, entre otros.
[28] OpenStreetMap: Es un proyecto colaborativo para crear un mapa editable del mundo que sea gratuito.

Para utilizar esta librería en una aplicación desarrollada con ASP.NET MVC, es necesario modificar la Vista que contendrá el mapa. Así, para añadir un mapa, debe crearse una instancia del mismo y configurar ciertos valores, tal como se indica en el Código 1.11. La línea 123 instancia un objeto del mapa y coloca el centro del mapa en las coordenadas [0,0] donde el primer valor corresponde a la latitud y el segundo valor a la longitud. También se coloca un nivel de acercamiento de 2. Las líneas 124, 125 y 126 agregan la capa del mapa que se quiere desplegar, que se obtiene de OpenStreetMap. Las líneas 127, 128 y 129 configuran parámetros opcionales. La línea 130 asigna la instancia a la variable creada para su uso.

```
123  var mymap = L.map('map').setView([0, 0], 2);
124  L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_token=pk.eyJ1IjoiamFhbjkwIiwiYSI6ImNqczYyeTBwZzBod2E0YW1zNWxlZDUxeWkifQ.73osagGl_LLhZT7M1aGvTA', {
125      attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, <a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>,
126      Imagery © < a href = "https://www.mapbox.com/" > Mapbox</a > ',
127      maxZoom: 18,
128      id: 'mapbox.streets',
129      accessToken: 'pk.eyJ1IjoiamFhbjkwIiwiYSI6ImNqczYyeTBwZzBod2E0YW1zNWxlZDUxeWkifQ.73osagGl_LLhZT7M1aGvTA'
130  }).addTo(mymap);
```

Código 1.11 Implementación de un mapa interactivo con Leaflet

GeoJSON es un formato para codificar una variedad de estructuras de datos geográficos [39].

GeoJSON es un formato de intercambio de datos geoespaciales basado en JSON. Define varios tipos de objetos y la manera en que se pueden combinar para representar datos sobre las características geográficas, sus propiedades y sus extensiones espaciales.

GeoJSON utiliza el sistema de referencia geográfico WGS84[29] y unidades de grados decimales [40].

GeoJSON está estandarizado por la IETF[30] en el RFC[31] 7946, el cual indica que un objeto GeoJSON puede representar una región del espacio, conocida como `Geometry`; un objeto limitado espacialmente, llamado `Feature`, que consiste en un arreglo de cero o más

[29] WGS84 (*World Geodetic System 1984*): Es un sistema de coordenadas geográficas en el cual se puede representar cualquier punto en la Tierra a través de tres unidades (x,y,z).
[30] IETF (*Internet Engineering Task Force*): Es una organización internacional de normalización que desarrolla y promueve estándares de Internet.
[31] RFC (*Request for Comments*): Son las publicaciones realizadas por la IETF. En ellos se describen protocolos, procedimientos y comentarios e ideas sobre los mismos.

objetos `Geometry`; y una lista de objetos `Feature`, denominada `FeatureCollection` [40].

Los objetos `Geometry` representan puntos, curvas y superficies. Los tipos `Geometry` soportados por GeoJSON son: `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` y `GeometryCollection`.

En el Código 1.12 se muestra un ejemplo de objeto `Point` definido utilizando GeoJSON. La línea 3 indica justamente el tipo de objeto que se describe (`Point`). La línea 4 especifica las coordenadas cartesianas de la ubicación del objeto `Point`.

```
2   {
3      "type": "Point",
4      "coordinates": [100.0, 0.0]
5   };
```

Código 1.12 Ejemplo de objeto `Point` en GeoJSON

GeoJSON puede ser utilizado en un mapa interactivo implementado con Leaflet, para asignar marcadores dentro del mismo, como puntos que corresponden a objetos GeoJSON.

Esto se muestra en el Código 1.13, donde se puede observar que inicialmente, se define el objeto `Point` y se lo asigna a la variable `geojsonPoint` definida (líneas 132, 133 y 134).

Los objetos GeoJSON son añadidos al mapa a través de una capa GeoJSON, se usa el objeto creado para agregarlo al mapa (línea 136) que previamente debió implementarse utilizando la librería Leaflet.

```
132   var geojsonPoint = {
133      "type": "Point",
134      "coordinates": [100.0, 0.0]
135   };
136   L.geoJSON(geojsonPoint).addTo(map);
```

Código 1.13 Utilización de GeoJSON en un mapa

1.3.7 KANBAN

Kanban es una metodología de desarrollo de software que prioriza la programación del trabajo a realizar para optimizar el tiempo de entrega de un producto [41].

Kanban permite a un equipo de trabajo o desarrollador gestionar de manera visual las tareas que se deben realizar, manteniendo el enfoque en la secuencia que sigue cada una de dichas tareas. Para esto se utiliza un sistema de tarjetas agrupadas en lo que se conoce como Tablero Kanban.

Kanban permite entender de forma visual el trabajo que se está realizando, y, con base en ese entendimiento, empezar a mejorar [42].

En la metodología Kanban, cada tarea o módulo a desarrollar como parte de un sistema es representado por una tarjeta. Las tarjetas se colocan en estados de flujo de trabajo. Estos corresponden a columnas dentro del tablero Kanban y representan el estado en el que se encuentra dicha tarea.

Un ejemplo de Tablero Kanban se presenta en la Figura 1.14.

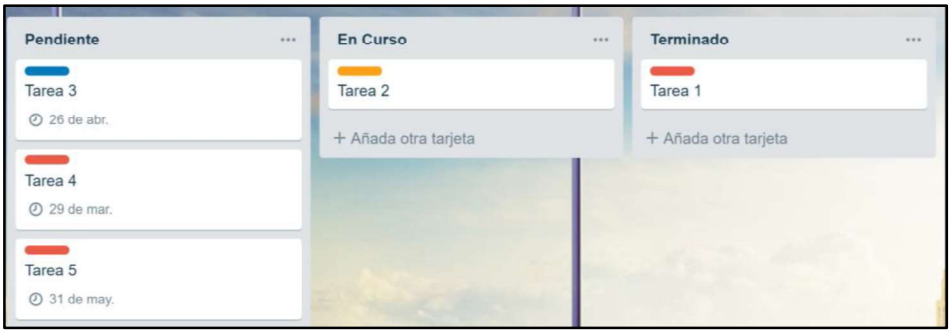


Figura 1.14 Ejemplo de Tablero Kanban

Kanban es una metodología flexible que se enfoca en la mejora continua.

Los estados de flujo de trabajo no son fijos y corresponden a las necesidades específicas de un grupo de trabajo o de un desarrollador, siempre teniendo en cuenta que, a menor cantidad, es mejor. De esta manera, por lo general se manejan tres estados de flujo de trabajo: Pendiente, En Curso y Terminado.

El estado Pendiente agrupa las tareas que aún faltan por hacer.

El estado En Curso congrega las tareas que actualmente se están desarrollando.

El estado Terminado contiene todo lo que se ha finalizado de desarrollar.

Cada una de las tareas itera a través de los estados de flujo de trabajo hasta que la totalidad de las mismas esté terminada.

Todas las tareas tienen aproximadamente el mismo tamaño o complejidad. El sistema es dividido en pequeñas tareas, lo que permite mejorar la capacidad de respuesta al cambio.

Entre las características de la metodología Kanban, se pueden mencionar [43]:

- Maximiza la productividad reduciendo el tiempo de inactividad, ya que todo está debidamente programado.
- Se enfoca en la entrega continua de pequeñas partes del software en iteraciones sucesivas, en lugar de grandes funcionalidades en bloques más grandes.
- Es flexible, puesto que permite reevaluar las prioridades constantemente y realizar cambios a tiempo [44].

2. METODOLOGÍA

En el presente Trabajo de Titulación se utilizará una investigación aplicada, cuyo objetivo es plasmar de forma práctica los conocimientos adquiridos en campos como bases de datos y aplicaciones web. De esta manera, se implementará un sistema que permita la administración de datos de glaciares, así como la visualización de los mismos mediante la generación de gráficas de series temporales.

Para recolectar información acerca de los requerimientos del sistema, se iniciará con la elaboración de entrevistas, mismas que serán aplicadas a los miembros del proyecto LMI GREAT ICE. También se mantendrán reuniones periódicas con los miembros del proyecto. Asimismo, se revisará el funcionamiento de la plataforma *National River Flow Archive* (NRFA) [45], para tomar como ejemplo las distintas opciones e interacciones con el usuario que se podrían implementar, considerando las particularidades propias del sistema a desarrollar. Con todo esto se elaborará una lista de los requerimientos funcionales y no funcionales del sistema.

En el desarrollo del sistema propuesto se utilizará la metodología de desarrollo ágil Kanban, debido a su flexibilidad y facilidad de uso, por lo que se establecerán las tareas a realizar con base en los requerimientos y se colocarán en un Tablero Kanban para su visualización. Este tablero contará con 4 columnas: Entrada, Implementación, Pruebas y Salida. La columna Entrada contendrá los distintos módulos que se desarrollarán en el sistema. La columna Implementación definirá todas las tareas necesarias para cumplir con la construcción de los módulos requeridos. La columna Pruebas mostrará las tareas esenciales para validar el funcionamiento de los módulos. Finalmente, la columna Salida contendrá los entregables de cada uno de los módulos del sistema.

El diseño del sistema se realizará en dos partes: base de datos y aplicación web. Para la base de datos se elaborará el diagrama entidad-relación y el diagrama relacional, que permitirán definir claramente las entidades, atributos y relaciones que formarán parte del sistema. En el caso de la aplicación web, se construirá el diagrama de clases que regirá el funcionamiento de la misma. También se elaborarán los *sketches* y *wireframes* que permitirán tener una idea clara de las interfaces de usuario de la aplicación web y el flujo entre ellas.

Por su parte, la implementación se realizará en 3 fases. En la primera fase se efectuará un procesamiento de la información provista por el proyecto LMI GREAT ICE, y en caso de ser necesario, se procederá con la depuración de la misma. Para la segunda fase, correspondiente a la construcción de la base de datos, se utilizará SQL Server

Management Studio 2017 y se codificarán *scripts* en SQL para la creación y población de tablas. En la tercera fase, que implica el desarrollo de la aplicación web, se hará uso del *framework* ASP.NET MVC 5 y del lenguaje de programación C#. Además, se utilizarán otras librerías que permitirán complementar el funcionamiento de la aplicación, como Leaflet.

2.1 ENTREVISTAS

Mediante la aplicación de entrevistas, se busca determinar los requerimientos funcionales y no funcionales de la aplicación web.

En total se aplicaron 11 entrevistas en inglés a miembros de LMI GREAT ICE, cuyo formato y resultados obtenidos se pueden encontrar de manera detallada en el ANEXO A. Sin embargo, a continuación, en la Tabla 2.1 y en la Tabla 2.2 se muestra un resumen de dichos resultados:

Tabla 2.1 Resultados de las entrevistas aplicadas (parte 1 de 2)

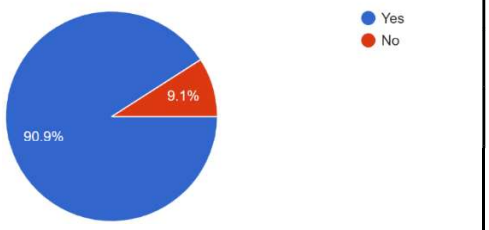
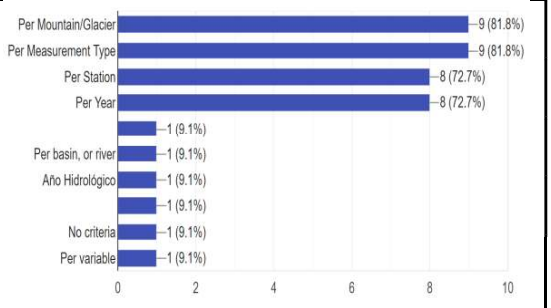
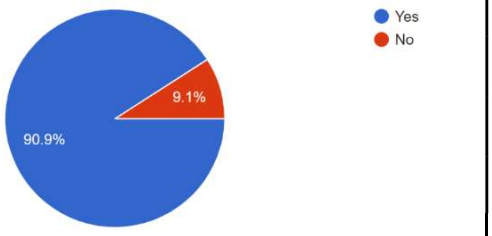
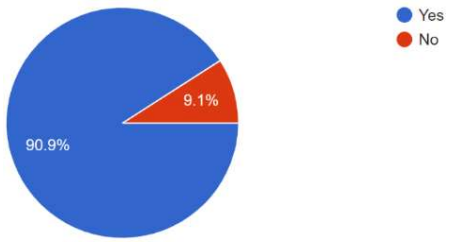
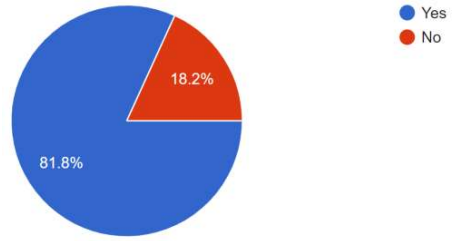
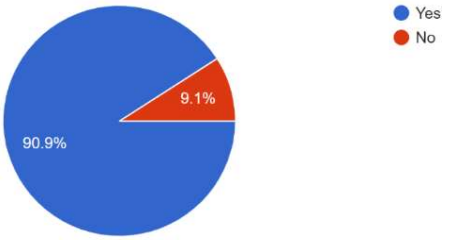
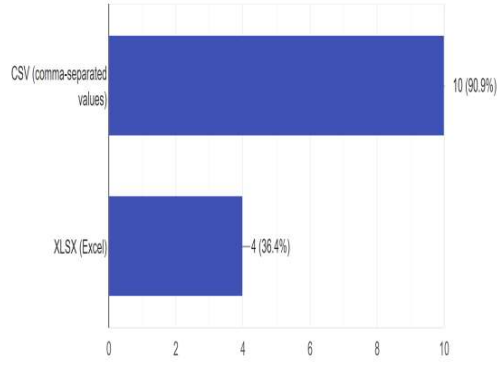
No	Pregunta	Resultados (%)
1	Would you like this web application to have different options for searching data?	
2	If your answer to the previous question was "Yes", what kind of searching criteria would you like this web application to have? (Please select one or more options)	
3	Besides the searching filters, would you like to have a map that allows you to select the measuring station? (Go to the following link as an example: http://nrfa.ceh.ac.uk/data/search)	

Tabla 2.2 Resultados de las entrevistas aplicadas (parte 2 de 2)

No	Pregunta	Resultados (%)
4	If your answer to the previous question was "Yes", would you like this web application to display the information associated with the station selected?	
5	Would you like the web application to present graphs of temporal lines of the daily data? (Go to the following link as an example: http://nrfa.ceh.ac.uk/data/station/meanflow/1001)	
6	Would you like the web application to allow download data?	
7	What file format do you prefer to download data? (Please select one or more options)	

2.2 REQUERIMIENTOS

En un inicio, el establecimiento de requerimientos funcionales y no funcionales se realizó con base en los resultados de las entrevistas aplicadas a los miembros de LMI GREAT ICE, considerando aquellas respuestas que obtuvieron una mayoría significativa de apoyo.

Luego, se tomaron en cuenta las peticiones directas de los miembros del proyecto que se solicitaron en las reuniones mantenidas.

Esto es posible gracias a que Kanban es una metodología ágil que permite incluir requerimientos conforme avanza el desarrollo del proyecto. Es decir, se pueden implementar cambios sin restricción durante la elaboración del sistema.

Finalmente, se consideraron aspectos de la plataforma *National River Flow Archive* (NRFA) [45] que podrían ayudar en la implementación del sistema.

2.2.1 REQUERIMIENTOS FUNCIONALES

- Los usuarios de la aplicación web podrán realizar búsquedas de información en la plataforma de manera libre. Es decir, para esta funcionalidad no es necesario contar con cuentas de usuario.
- Para la carga y descarga de datos, el usuario deberá crear una cuenta en la aplicación web. El nivel de acceso a la información será determinado por los administradores del sistema.
- Existirán tres perfiles de usuario: `Administrator`, `Download&Upload` y `Download`. Cada perfil tiene distintos privilegios de acceso al sistema.
- Los usuarios con perfil `Administrator` serán capaces de añadir, eliminar o editar información de elementos como: usuarios, montañas, glaciares, estaciones, instituciones y países, así como la opción para cargar nuevos documentos y descargar información del sistema. También, se podrá mostrar un listado con los distintos registros de dichos elementos. Previamente estarán registrados dos usuarios con este perfil.
- El usuario con perfil `Download&Upload` podrá cargar nuevos archivos a la base de datos, así como también descargar información existente.
- El usuario con perfil `Download` únicamente podrá descargar datos desde la base de datos.
- Para crear una cuenta, el usuario deberá ingresar información personal como nombre, correo, contraseña, género, título, fecha de nacimiento, país e institución. Esta

información será enviada por correo electrónico a los administradores del sistema, quienes decidirán si aprueban o no la creación de dicha cuenta.

- Mientras los administradores no aprueben la creación de la nueva cuenta de usuario, esta es guardada en la base de datos con el perfil `Download`. Cuando los administradores aprueben la creación de la cuenta, ellos le asignarán el perfil que consideren pertinente.
- Si los administradores del sistema aceptan la creación de una cuenta, procederán a la activación de la misma y le asignarán uno de los perfiles de usuario antes mencionados.
- Si los administradores del sistema no aprueban la creación de la cuenta de un usuario específico, su cuenta permanecerá inactiva y dicho usuario no podrá iniciar sesión en el sistema hasta que su cuenta sea activada.
- Para iniciar sesión, un usuario deberá ingresar su correo electrónico registrado, así como la contraseña correspondiente. Si los datos ingresados son correctos, el inicio de sesión será válido. Caso contrario, se mostrará un mensaje indicando que las credenciales ingresadas son incorrectas y no se permitirá iniciar sesión en el sistema.
- Para crear nuevos registros de montañas, los administradores del sistema deberán ingresar el nombre de la montaña, el país al que pertenece y un estado del elemento (activo o inactivo).
- Para crear nuevos registros de glaciares, los administradores del sistema deberán ingresar el nombre del glaciar, la montaña a la que pertenece y un estado del elemento (activo o inactivo).
- Para crear nuevos registros de estaciones, los administradores del sistema deberán ingresar el nombre de la estación, latitud, longitud, altitud, glaciar en el que se encuentra, ubicación (dentro o fuera del glaciar), tipo de medida del que se encarga la estación (meteorología, glaciología o hidrología) y un estado del elemento (activo o inactivo).
- Para crear nuevos registros de instituciones, los administradores del sistema deberán ingresar el nombre de la nueva institución.
- Para cargar un nuevo documento, las personas autorizadas deberán ingresar la fecha de inicio de medición que consta en el documento, fecha de finalización, tipo de dato a ingresar (Observación o Simulación), tipo de medida (meteorología, glaciología o

hidrología), estación a la que corresponde el documento y adjuntar el documento que se desea subir a la plataforma.

- El documento original se cargará en una carpeta y, adicionalmente, la información contenida en el documento se cargará a las tablas respectivas en la base de datos.
- Dependiendo del tipo de medida del documento que se desea cargar, se realizará una validación en cuanto al formato del mismo. Para ello, se definirán previamente formatos de documento que tomen en consideración el tipo de medida que se desea almacenar en el sistema.
- Se deberá registrar los usuarios que suben cada documento para tener una constancia de quién lo hace.
- Al eliminar un elemento, no se borrará de la base de datos. En su lugar, los elementos se pondrán en un estado inactivo que impida que se utilicen, exceptuando los documentos, que si deberán ser eliminados de la base de datos.
- Los valores de glaciología, meteorología o hidrología contenidos en los documentos si deberán eliminarse de la base de datos cuando se lo requiera.
- Los usuarios podrán restablecer la contraseña de sus cuentas en caso de que no la recuerden. Para esto, se generará una clave temporal aleatoria que será enviada al correo electrónico del usuario, con la cual podrá entrar al sistema y cambiar su contraseña.
- En la página principal de la aplicación web, se mostrarán los filtros para realizar búsquedas de información, así como un mapa con los marcadores que representan a las estaciones que actualmente tiene el proyecto de investigación LMI GREAT ICE. También se mostrarán las opciones para iniciar sesión y crear una nueva cuenta de usuario.
- Para realizar una búsqueda de información, los usuarios del sistema deberán especificar el país, montaña, glaciar, tipo de dato, tipo de medida, estación, variable y año. Los dos últimos criterios dependerán del tipo de medida seleccionado con anterioridad.
- Los resultados de las búsquedas de información realizadas serán presentados en forma de gráficas temporales. Los valores resultantes podrán ser descargados en formato CSV (*comma separated-values*) por los usuarios que han iniciado sesión.

- Conjuntamente con los resultados de la búsqueda se mostrarán datos relacionados a la estación que tomó las medidas de la información buscada.
- Los marcadores en el mapa estarán asociados a las estaciones, y al dar clic en alguno de ellos, se desplegarán los documentos cargados en el sistema asociados a la estación seleccionada.

2.2.2 REQUERIMIENTOS NO FUNCIONALES

- La información gestionada por el sistema será almacenada en una base de datos relacional.
- La contraseña de los usuarios será cifrada previo a su almacenamiento en la base de datos.
- La aplicación web será desarrollada en ASP.NET MVC.
- El mapa interactivo deberá ser implementado utilizando una librería de código abierto y gratuita.

2.3 TABLERO KANBAN

Para la realización del tablero Kanban, primero deben establecerse las tareas a cumplir. Estas tareas se han dividido en tres grupos principales: Funcionalidad Básica, Validaciones y Gráficos. En Funcionalidad Básica se encuentran las tareas que permitirán realizar operaciones de inserción, modificación, lectura y eliminación de elementos en sus distintos tipos. También aquellos métodos que realicen cálculos, búsquedas y, carga y descarga de datos. Este grupo permitirá cumplir, en su mayoría, con los requisitos funcionales descritos anteriormente. Asimismo, se encuentran las tareas para la creación de la base de datos y su conexión con la aplicación web.

El grupo de Validaciones congrega las tareas que de alguna manera revisan la parte funcional. Tal es el caso de validaciones de formato de documentos, de tipos de datos, estructura, etc. Finalmente, Gráficos agrupa las tareas que tienen que ver con la generación de las gráficas temporales, el mapa y su interacción.

En el tablero Kanban se contará con 4 columnas: Entrada, Implementación, Pruebas y Salida. La Tabla 2.3 y la Tabla 2.4 muestran el tablero Kanban correspondiente a las tareas del grupo de Funcionalidad Básica:

Tabla 2.3 Tablero Kanban de Funcionalidad Básica (parte 1 de 2)

Funcionalidad Básica			
Entrada	**Implementación**	**Pruebas**	**Salida**
Creación de la Base de Datos	Diseño del diagrama entidad-relación de la base de datos. Codificación de los scripts de creación de la base datos con base en el diagrama diseñado.	N/A	Base de datos
Creación de la aplicación web y conexión con la base de datos a través de Entity Framework	Creación de un nuevo proyecto en Visual Studio 2017 de tipo Aplicación Web ASP.NET (.NET Framework). Selección de la plantilla MVC para ASP.NET. Creación del modelo de datos mediante Entity Framework, que utilizará la base de datos creada para generar las clases necesarias en la aplicación.	Comprobar la conexión con la base de datos.	Aplicación web conectada a la base de datos
Administración de usuarios	Codificación de los métodos necesarios para Crear, Leer, Editar y Eliminar usuarios dentro de la aplicación web.	Corroborar si los métodos creados modifican o leen la base de datos efectivamente.	Administración de usuarios
Envío de correo de confirmación/aprobación	Codificación del método requerido para el envío de correo al administrador del sistema con la información del usuario que desea crear una cuenta.	Comprobar si al crear un nuevo usuario, su información es enviada al correo electrónico del administrador del sistema.	Envío de correo de confirmación/aprobación
Acceso al sistema	Codificación de los métodos necesarios para iniciar sesión en la aplicación y acceder a la misma.	Corroborar si el inicio de sesión funciona correctamente.	Acceso al sistema

Tabla 2.4 Tablero Kanban de Funcionalidad Básica (parte 2 de 2)

Funcionalidad Básica			
Entrada	**Implementación**	**Pruebas**	**Salida**
Recuperación de contraseña	Implementación del código necesario para el método que enviará un correo al usuario para recuperar su contraseña.	Comprobar si el envío del correo para recuperar contraseña funciona.	Recuperación de contraseña
Administración de montañas	Codificación de los métodos necesarios para Crear, Leer, Editar y Eliminar montañas dentro de la aplicación web.	Corroborar si los métodos creados modifican o leen la base de datos efectivamente.	Administración de montañas
Administración de glaciares	Realizar la codificación de los métodos necesarios para Crear, Leer, Editar y Eliminar glaciares dentro de la aplicación web.	Corroborar si los métodos creados modifican o leen la base de datos efectivamente.	Administración de glaciares
Administración de estaciones	Codificación de los métodos necesarios para Crear, Leer, Editar y Eliminar estaciones dentro de la aplicación web.	Corroborar si los métodos creados modifican o leen la base de datos efectivamente.	Administración de estaciones
Administración de instituciones	Implementación del código necesario para Crear, Leer y Editar instituciones dentro de la aplicación web.	Corroborar si los métodos creados modifican o leen la base de datos efectivamente.	Administración de instituciones
Carga de documentos	Codificación de los métodos necesarios para subir nuevos documentos con información de glaciares.	Corroborar si los métodos creados funcionan correctamente.	Carga de documentos
Descarga de información	Realizar la codificación de los métodos necesarios para realizar la descarga de la información solicitada.	Comprobar si la descarga funciona correctamente.	Descarga de información
Búsqueda de información	Implementación del código necesario para la búsqueda de información en base a los criterios solicitados en la entrevista de requerimientos.	Comprobar si la búsqueda devuelve los resultados deseados	Búsqueda de información

La Tabla 2.5 muestra el tablero Kanban correspondiente al grupo de Validaciones. Aquí se agrupan tareas para verificar que se cumplan ciertas restricciones en función de la tarea a realizar.

Por su parte, la Tabla 2.6 presenta el tablero Kanban del grupo de Gráficos, en el cual se agrupan las tareas necesarias para generar los gráficos de series temporales, implementación del mapa interactivo y la colocación de marcadores en el mismo, que representen las estaciones que recolectan información para el proyecto LMI GREAT ICE.

Tabla 2.5 Tablero Kanban de Validaciones

Validaciones			
Entrada	**Implementación**	**Pruebas**	**Salida**
Descarga de archivos solo para usuarios registrados y activos	Codificar las validaciones necesarias para que la descarga de información la puedan realizar solo los usuarios registrados y con estado Activo.	Comprobar los permisos de descarga	Descarga de archivos solo para usuarios registrados
Recuperación de contraseña solo para usuarios registrados y activos	Codificar las validaciones necesarias para que la recuperación de contraseñas la puedan realizar solo los usuarios registrados y con estado Activo.	Comprobar la recuperación de contraseñas	Recuperación de contraseña solo para usuarios registrados y activos
Inicio de sesión solo para usuarios válidos	Realizar la codificación de los métodos necesarios para que únicamente los usuarios registrados y con estado Activo puedan iniciar sesión.	Comprobar inicio de sesión	Inicio de sesión solo para usuarios válidos
Carga de documentos con formato CSV	Realizar la codificación del método que permitirá subir nuevos documentos únicamente si tienen formato CSV.	Comprobar validación del tipo de formato CSV	Carga de documentos con formato CSV
Validación de estructura de documentos con información de Meteorología	Implementación del código requerido para validar la estructura de documentos de Meteorología y guardarlos.	Cargar un documento de Meteorología	Estructura de documentos de Meteorología validada
Validación de estructura de documentos con información de Glaciología	Implementación del código requerido para validar la estructura de documentos de Glaciología y guardarlos.	Cargar un documento de Glaciología	Estructura de documentos de Glaciología validada

Tabla 2.6 Tablero Kanban de Gráficos

Gráficos			
Entrada	**Implementación**	**Pruebas**	**Salida**
Gráficas Temporales	Codificación de los métodos requeridos para que se generen las gráficas temporales de acuerdo a los criterios de búsqueda seleccionados.	Realizar una búsqueda y verificar si los resultados obtenidos en la gráfica temporal coinciden con lo deseado.	Gráficas Temporales
Mapa	Realizar la codificación necesaria para generar un mapa utilizando la librería Leaflet.	Visualizar el mapa creado.	Mapa
Marcadores en el mapa	Implementación del código necesario para generar los marcadores correspondientes a las estaciones y visualizarlos en el mapa.	Visualizar los marcadores de estaciones.	Marcadores en el mapa

2.4 DIAGRAMA ENTIDAD-RELACIÓN

El diagrama entidad-relación permitirá esquematizar el modelo utilizado para el diseño de la base de datos. Y, como se mencionó anteriormente, se partirá de la base de datos para posteriormente, mediante Entity Framework, generar las clases necesarias. Así, se han definido las entidades a utilizar, los atributos de dichas entidades y las relaciones entre ellas. Para la explicación, el diagrama entidad-relación se ha dividido en 3 partes, puesto que existen 19 entidades, algunas de las cuales tienen muchos atributos. Esto se muestra en la Figura 2.1, la Figura 2.2 y la Figura 2.3. La entidad común entre las tres figuras es `enteredData`.

2.5 DIAGRAMA RELACIONAL

En la Figura 2.4 se presenta el diagrama relacional utilizado para generar la base de datos. Para el diseño de la base de datos se han tomado en cuenta los requerimientos funcionales y no funcionales anteriormente descritos, obteniendo las siguientes tablas:

- `userState`: Esta tabla almacena los estados del usuario que tiene una cuenta en el sistema. Así, se tienen tres estados: `Active`, `Inactive` y `Pending`. En el estado `Active`, los usuarios pueden ejecutar todas las tareas que su perfil le permiten; los

usuarios adquieren este estado una vez que los administradores del sistema aprueban la creación de su cuenta.

El estado `Inactive` corresponde a los usuarios que se eliminan del sistema, pero por cuestiones de consistencia, no son eliminados de la base de datos; en este estado los usuarios no pueden ejecutar ninguna acción. Finalmente, el estado `Pending` corresponde a aquellos usuarios que han creado su cuenta, pero aún falta la aprobación de los administradores del sistema para la activación de la misma; en este estado los usuarios no pueden ejecutar acciones.

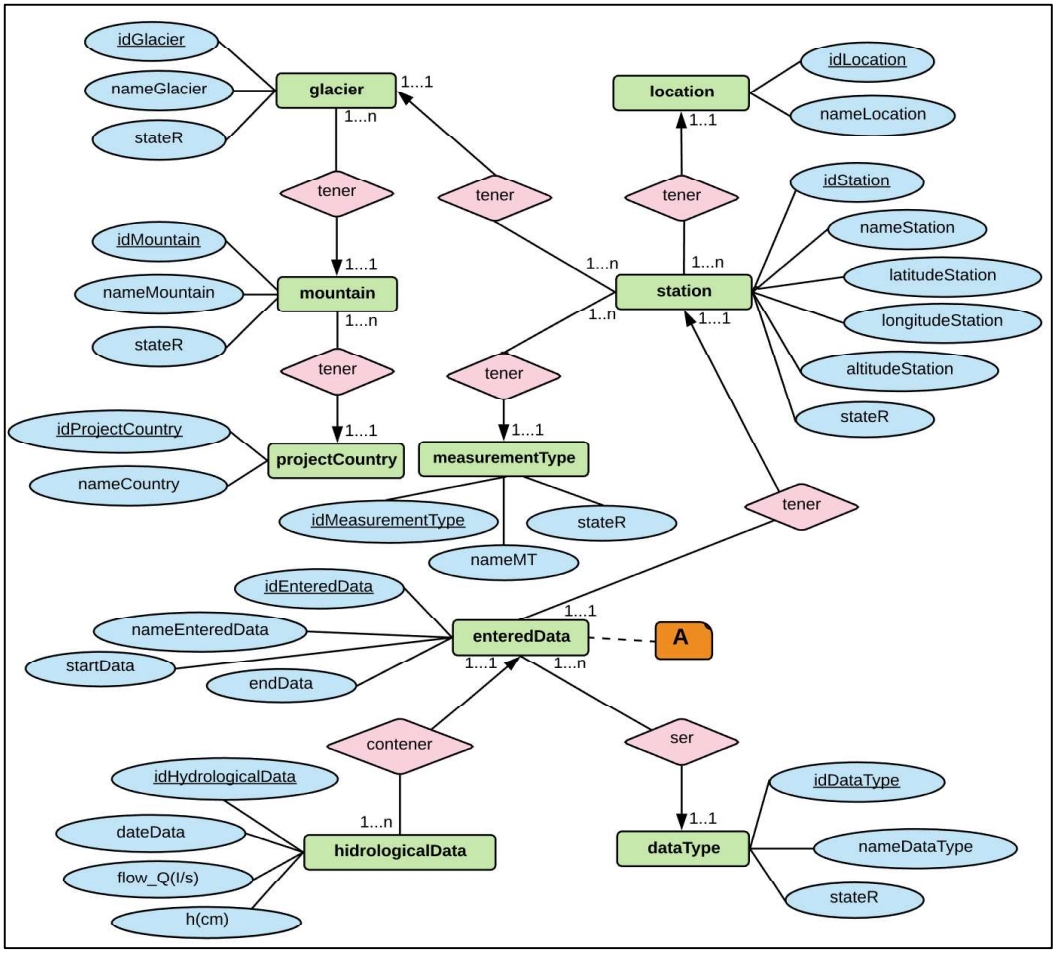


Figura 2.1 Diagrama entidad – relación (parte 1 de 3)

- `systemUser`: En esta tabla se almacenan todos los atributos de un usuario del sistema, tales como datos personales, institución, estado y perfil asignado.

- `gender`: Esta table fue creada de forma complementaria para almacenar géneros. Está relacionada a la tabla `systemUser` de tal forma que una persona solo puede tener un género, pero un mismo género puede ser utilizado por varias personas.

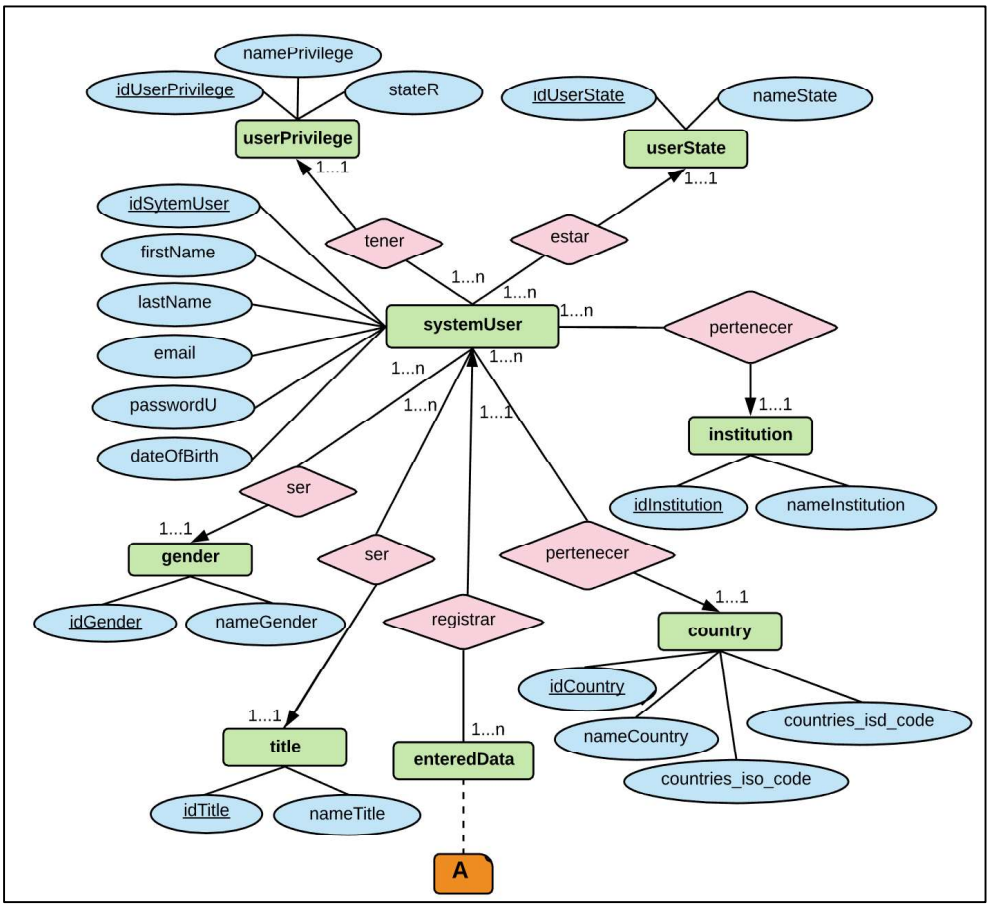


Figura 2.2 Diagrama entidad – relación (parte 2 de 3)

- `userPrivilege`: En esta tabla se almacenan los perfiles que se les asignará a cada usuario con una cuenta. `Administrator` tiene permisos ilimitados sobre el sistema, esto incluye crear, eliminar, editar cualquier entidad que permita esas acciones. También, pueden descargar y cargar información al sistema. Los usuarios con perfil `Download&Upload` pueden descargar información y también subir nuevos documentos con información meteorológica, glaciológica o hidrológica al sistema. Los usuarios con perfil `Download` únicamente pueden descargar información del sistema. Al crearse una nueva cuenta de usuario, esta se guarda en la base de datos con el

atributo `userState` igual a `Pending` y el atributo `userPrivilege` igual a `Download` por defecto. Esto hasta que los administradores del sistema acepten la creación de la misma y le asignen el perfil que ellos decidan.

- `country`: Esta tabla se creó para que almacene información de los países del mundo y, que los usuarios nuevos, al crear su cuenta, seleccionen el país al que pertenecen.

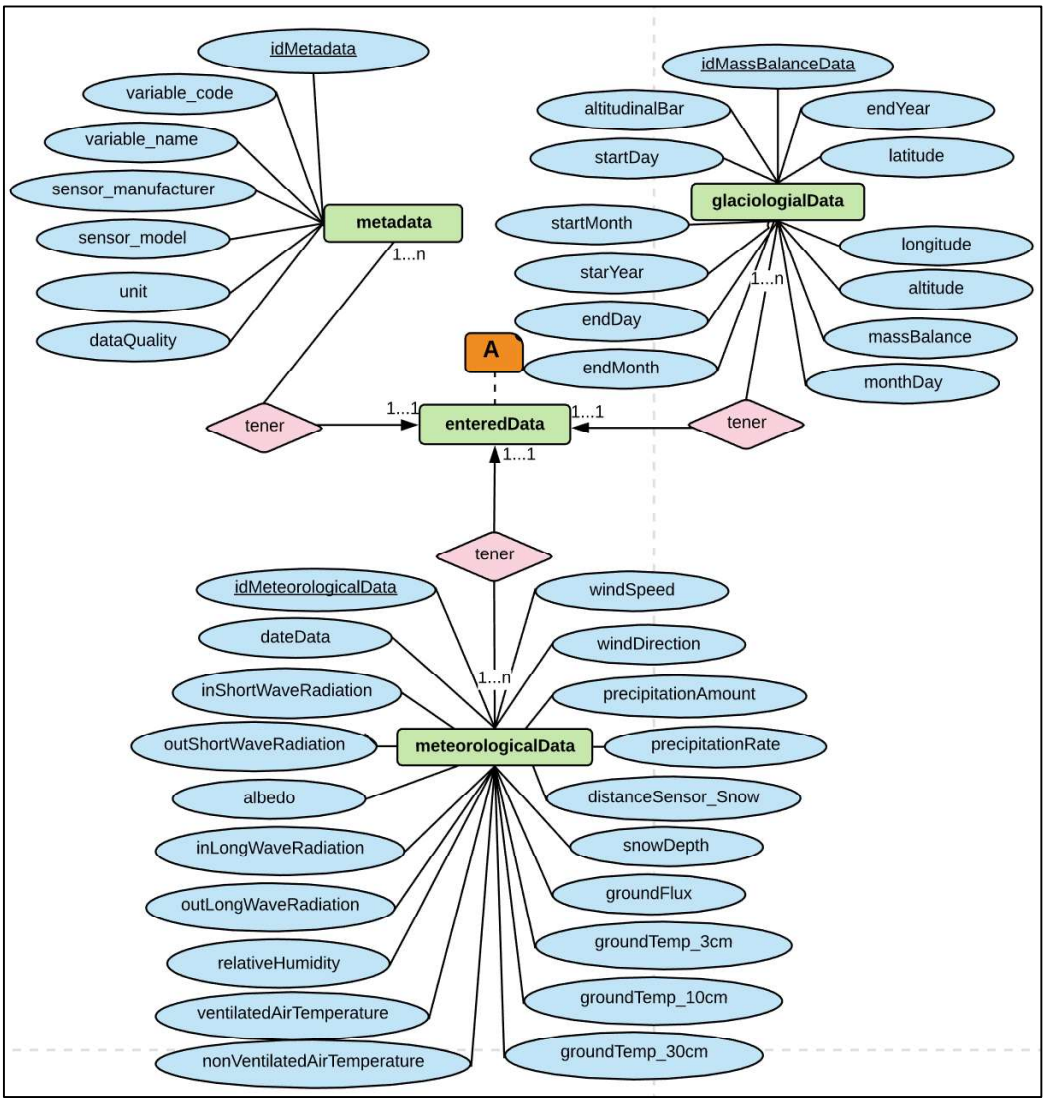


Figura 2.3 Diagrama entidad – relación (parte 3 de 3)

- `institution`: En esta tabla se almacena información sobre las instituciones que forman parte del proyecto de investigación LMI GREAT ICE. Entre ellas, el IRD *(Institut de Recherche pour le Developpement)* y la Escuela Politécnica Nacional.

- `title`: Esta tabla contiene los títulos que una persona desea tener. `Mr`, `Ms`, `Doctor`, `Professor`.

- `glacier`: En esta tabla se almacena información sobre los glaciares y sus características, de los que actualmente el proyecto LMI GREAT ICE tiene datos, ya sean Meteorológicos, Glaciológicos o Hidrológicos. Actualmente, esta tabla contiene información de los glaciares `ANTISANA-15` (Ecuador) y `ZONGO` (Bolivia).

- `mountain`: Esta tabla tiene relación directa con la tabla `glacier`. Almacena información sobre las montañas que contienen a los glaciares que forman parte del proyecto LMI GREAT ICE. Así, una montaña puede tener varios glaciares, pero un glaciar solo puede pertenecer a una montaña.

- `projectCountry`: En esta tabla se guarda información sobre los países en los que actualmente el proyecto LMI GREAT ICE hace estudios. Por el momento son Ecuador y Bolivia.

- `station`: Contiene información sobre las estaciones que actualmente recopilan datos de los glaciares para el proyecto LMI GREAT ICE, y sus características. Actualmente hay siete estaciones, cuatro de Meteorología, dos de Glaciología y una de Hidrología.

- `location`: Esta tabla almacena información de la localización de una estación en un glaciar. Una estación puede estar ubicada dentro del glaciar (`inside glacier`) o fuera del glaciar (`outside glacier`).

- `dataType`: En esta tabla se guardan los tipos de datos a almacenar. Existen dos tipos de datos: `Observation` y `Simulation`.

  Actualmente solo funciona el tipo de dato `Observation`, que corresponden a los archivos con formato CSV que contienen información de los glaciares. Sin embargo, por petición de los miembros de LMI GREAT ICE, se agregó la opción `Simulation` para que en un futuro se puedan almacenar también archivos NetCDF con simulaciones sobre los glaciares.

- `measurementType`: Esta tabla los tipos de medida de los que se puede recopilar información sobre los glaciares.

  Existen tres tipos de medida, basados en los documentos otorgados por el proyecto LMI GREAT ICE: Meteorología (`Meteorological`), Glaciología (`Glaciological`) e Hidrología (`Hydrological`).

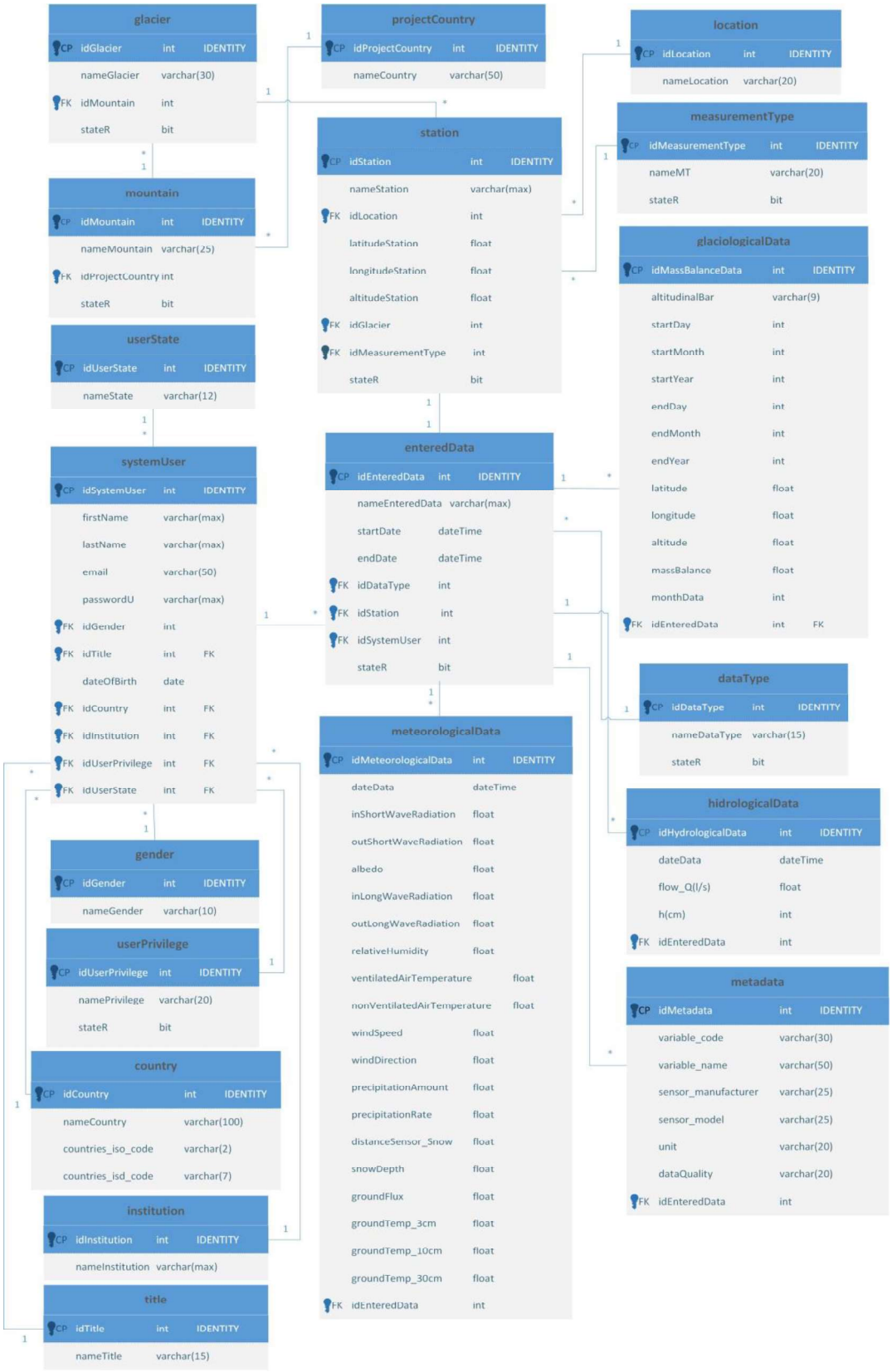


Figura 2.4 Diagrama relacional

- `enteredData`: Los elementos de esta tabla representan a cada uno de los documentos (archivos con formato CSV) con información sobre los glaciares, mas no a la información en sí. Contiene detalles que caracterizan al documento para manipular de manera más conveniente, tales como: nombre del documento, fecha en la que iniciaron mediciones, fecha en la que finalizan las mediciones, tipo de dato del documento, estación con la que se recolectó la información contenida en el documento, usuario que subió al sistema dicho documento, etc.

- `meteorologicalData`: En esta tabla se almacena la información de cada uno de los documentos de Meteorología. Es decir, se almacenan los valores contenidos dentro del archivo CSV y que corresponden a las mediciones realizadas para distintas variables como: `inShortWaveRadiation`, `outShortWaveRadiation`, `albedo`, `relativeHumidity`, entre otras. Esta tabla puede contener valores nulos debido a que, dependiendo de la localización de la estación (`Inside glacier/Outside glacier`), algunas variables pueden no ser medidas. Todas las tuplas tienen asignado un `id` del documento que las contiene (`enteredData`).

- `metadata`: Esta tabla fue creada debido a que los archivos de Meteorología contienen no solo los valores de las variables medidas (`meteorologicalData`), sino que también contienen metadatos sobre la estación con la que se recopiló dicha información. La tabla almacena esos metadatos asociados a cada estación.

- `glaciologicalData`: Esta tabla contiene los valores contenidos dentro de los archivos de Glaciología. La información almacenada es: `altitudinal bar, start date, end date, latitude, longitude, altitude, mass balance`, entre otros. Cada tupla tiene asignado un `id` del documento que la contiene (`enteredData`).

- `hydrologicalData`: En esta tabla se almacenan los valores contenidos dentro de los documentos de Hidrología. Por ejemplo, `date, flow, h` y el `id` del documento que las contiene (`enteredData`).

2.6 DIAGRAMA DE CLASES

El diagrama de clases describe la estructura de una aplicación web, y detalla las clases, atributos y métodos de cada una, y las relaciones entre clases.

Al utilizar ASP.NET MVC, el Modelo y los Controladores están representados por clases.

2.6.1 MODELO

En el Modelo, cada entidad o tabla de la base de datos está representada por una clase. Cada clase a su vez tiene varios atributos que corresponden a cada una de las columnas de su respectiva tabla. Esto se debe a que las clases del Modelo fueron generadas a partir de la base de datos mediante Entity Framework. Las Clases Entidad creadas automáticamente al generar el Modelo con Entity Framework son:

- `country.cs`: Esta clase representa a la tabla `country`. Es usada, por ejemplo, al desplegar una lista con nombres de países al crear una nueva cuenta de usuario.
- `gender.cs`: Representa a la tabla `gender`. Se utiliza, por ejemplo, al desplegar una lista con géneros al momento de crear una nueva cuenta de usuario.
- `institution.cs`: Esta clase representa a la tabla `institution`. Se usa, por ejemplo, al desplegar una lista con el nombre de las instituciones a las que puede pertenecer un usuario.
- `systemUser.cs`: Representa a la tabla `systemUser`. Es utilizada, por ejemplo, al listar los usuarios que están registrados en el sistema y su respectiva información.
- `title.cs`: Representa a la tabla `title`. Es usada, por ejemplo, al desplegar una lista con títulos honoríficos para que un usuario seleccione alguno al momento de crear una nueva cuenta.
- `userState.cs`: Representa a la tabla `userState`. Se utiliza, por ejemplo, al desplegar una lista para que el Administrador seleccione el estado de usuario que le asignará a una nueva cuenta creada.
- `userPrivilege.cs`: Representa a la tabla `userPrivilege`. Se usa, por ejemplo, al desplegar una lista con los perfiles de usuario para que el Administrador seleccione uno, el mismo que se le asignará a una nueva cuenta de usuario creada.
- `mountain.cs`: Representa a la tabla `mountain`. Es utilizada, por ejemplo, al desplegar una lista con los nombres de las montañas almacenadas en el sistema, que se muestran al momento de crear un nuevo registro de la entidad `glacier`.
- `glacier.cs`: Representa a la tabla `glacier`. Es usada, por ejemplo, al desplegar una lista con los nombres de los glaciares guardados en el sistema, que se muestran al momento de crear un nuevo registro de la entidad `station`.

- `enteredData.cs`: Representa a la tabla `enteredData`. Se utiliza, por ejemplo, al desplegar una lista con los registros de los documentos guardados en el sistema.
- `meteorologicalData.cs`: Representa a la tabla `meteorologicalData`. Se usa, por ejemplo, al acceder o guardar información de meteorología en la base de datos.
- `projectCountry.cs`: Representa a la tabla `projecCountry`. Es utilizada, por ejemplo, al desplegar una lista que contiene los nombre de países en los que trabaja actualmente el proyecto de investigación LMI GREAT ICE, que se muestran al momento de crear un nuevo registro de la entidad `mountain`.
- `station.cs`: Representa a la tabla `station`. Es usada, por ejemplo, al desplegar una lista con los nombres de las estaciones guardadas en el sistema, que se muestran al realizar una búsqueda de datos.
- `dataType.cs`: Representa a la tabla `dataType`. Se utiliza, por ejemplo, al desplegar una lista con los nombres de los tipos de datos (`Observation/Simulation`). Uno de los cuales debe ser seleccionado al cargar nuevos documentos al sistema.
- `hydrologicalData.cs`: Representa a la tabla `hydrologicalData`. Se usa, por ejemplo, al acceder o guardar información de hidrología en la base de datos.
- `location.cs`: Representa a la tabla `location`. Es utilizada, por ejemplo, al desplegar una lista con las localizaciones de las estaciones en un glaciar (`Inside glacier/Outside glacier`). Una de las cuales debe ser seleccionada al cargar nuevos documentos al sistema.
- `measurementType.cs`: Representa a la tabla `measurementType`. Es usada, por ejemplo, al listar los tipos de medida al momento de realizar una búsqueda de información.
- `metadata.cs`: Representa a la tabla `metadata`. Se utiliza, por ejemplo, al acceder o guardar información de los metadatos de las estaciones de meteorología en la base de datos.
- `glaciologicalData.cs`: Representa a la tabla `glaciologicalData`. Se usa, por ejemplo, al acceder o guardar información de glaciología en la base de datos.

Al generarse el Modelo, también se creó la Clase Contexto `DB_LMIGREATICEEntities` que permite establecer una sesión con la base de datos para acceder o manipular la información contenida en ella.

Adicionalmente, se crearon de forma manual las clases:

- `FilteredData`: Se usa al generar un resumen con la información buscada por el usuario para que sea descargada en formato CSV
- `Holder`: Se utiliza para llevar un control de los mensajes de error a desplegar.
- `StructureValidation`: Contiene métodos para realizar validaciones en la estructura de los documentos antes de subirlos a la base de datos
- `EnteredDataExt`: Permite pasar los datos ingresados por el usuario en la Vista de búsqueda al método respectivo. Hereda de la clase `enteredData` y contiene otros atributos propios.
- `enteredDataSimplify`: Se usa al obtener y desplegar una lista con información de los documentos (registros de `enteredData`) asociados a una estación en el mapa interactivo. La lista desplegada no contendrá todos los atributos de la clase `enteredData`.
- `StationTrick`: Se utiliza para devolver una lista de todas las estaciones guardadas en la base de datos y cuyo estado es `Active`.

La Figura 2.5 presenta el diagrama de clases del Modelo, utilizado en el desarrollo del sistema planteado. En él se pueden observar todas las clases que se detallaron anteriormente y la forma en que están asociadas.

La Figura 2.6 por su parte, indica la manera en que se relacionan las Clases Entidad con la Clase Contexto, todas generadas automáticamente con Entity Framework. Las Clases Entidad son representaciones de cada una de las tablas existentes en la base de datos.

La Clase Contexto es posiblemente la clase más importante del Modelo, pues con ella se puede acceder a la información de la base de datos o manipularla.

Como se puede observar, los atributos de la Clase Contexto son propiedades de tipo `DbSet<Entity>` de las clases creadas al generar el Modelo mediante Entity Framework.

Una clase se transforma en entidad cuando es incluida como una propiedad de tipo `DbSet<Entity>` en la Clase Contexto.

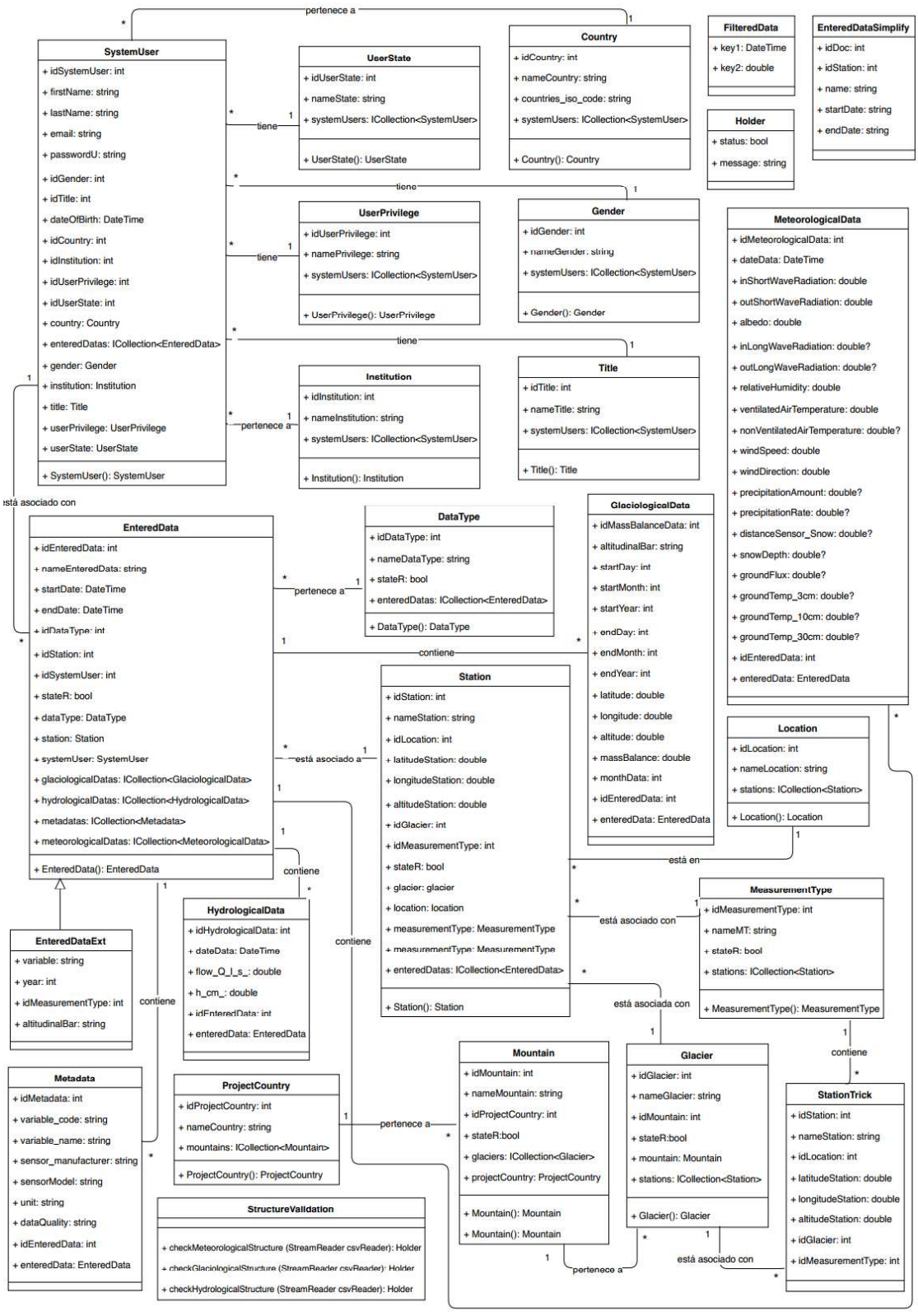


Figura 2.5 Diagrama de clases del Modelo

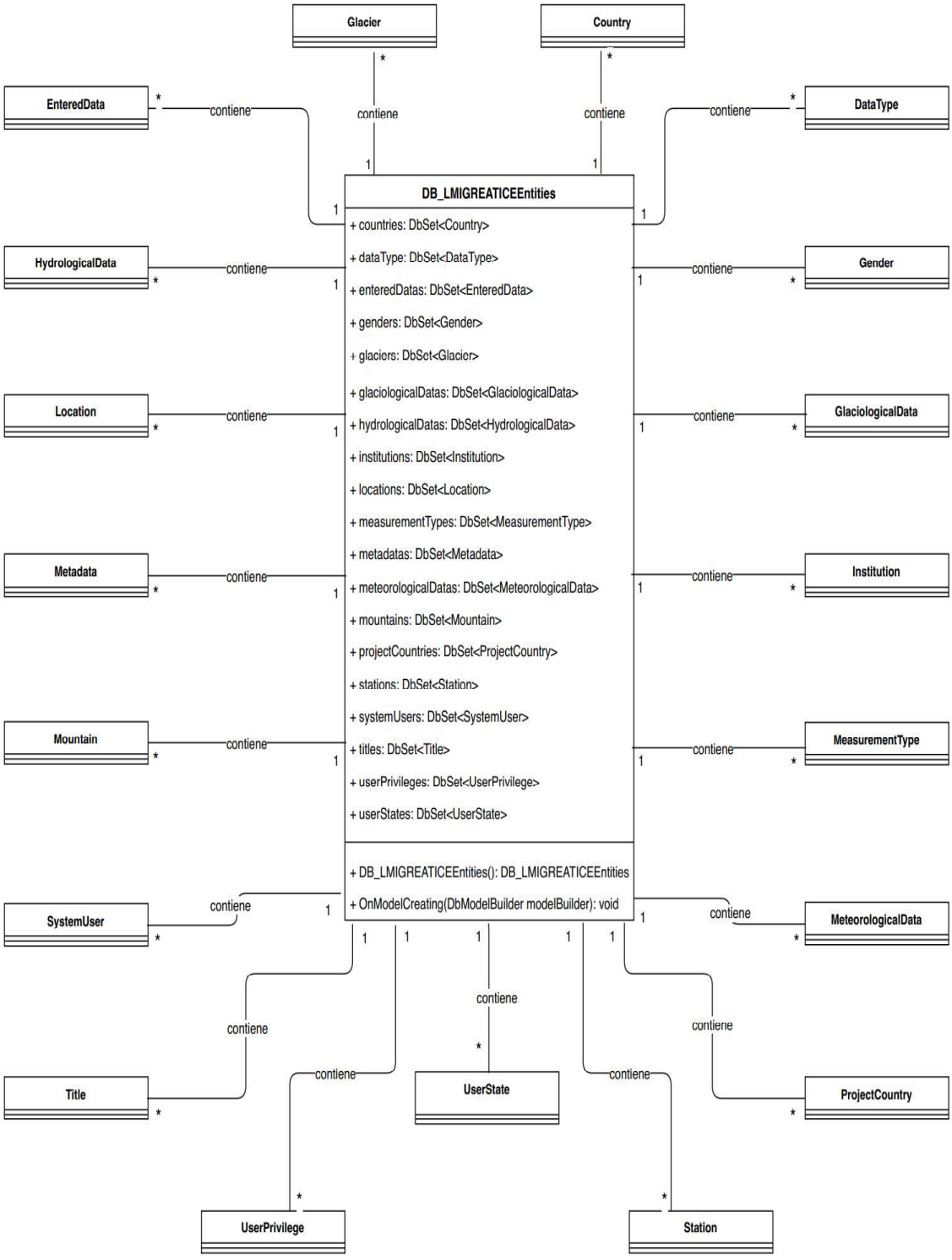


Figura 2.6 Relación entre las Clases Entidad y la Clase Contexto

Las propiedades `countries`, `dataTypes`, `enteredDatas`, `genders`, `glaciers`, `glaciologicalDatas`, `hydrologicalDatas`, `intitutions`, `locations`, `measurementTypes`, `metadatas`, `meteorologicalDatas`, `mountains`, `projectCountries`, `stations`, `systemUsers`, `userPrivileges` y `userStates` se denominan conjuntos de entidades.

Mientras que, `country`, `dataType`, `enteredData`, `gender`, `glacier`, `glaciologicalData`, `hydrologicalData`, `institution`, `location`, `measurementType`, `metadata`, `meteorologicalData`, `mountain` `projectCountry`, `station`, `systemUser`, `userPrivilege` y `userState` se denominan entidades.

2.6.2 CONTROLADOR

Cada Controlador creado en la aplicación web, está representado por una clase. En este caso, se dispone de los siguientes Controladores:

- `glaciersController.cs`: Contiene atributos y métodos para leer, añadir, editar o eliminar información de los glaciares estudiados por el proyecto LMI GREAT ICE.
- `institutionsController.cs`: Contiene atributos y métodos para leer, añadir o editar información de las instituciones que actualmente forman parte del proyecto LMI GREAT ICE.
- `mountainsController.cs`: Contiene atributos y métodos para leer, añadir, editar o eliminar información de las montañas que contienen a los glaciares estudiados.
- `stationsController.cs`: Contiene atributos y métodos para leer, añadir, editar o eliminar información de las estaciones que actualmente recogen datos de los glaciares.
- `systemUsersController.cs`: Contiene atributos y métodos para leer, añadir, editar o eliminar información de los usuarios registrados en el sistema. También tiene métodos para iniciar sesión, cerrar sesión, recuperar la contraseña, enviar correo de confirmación y cifrar la contraseña.
- `enteredDataController.cs`: Contiene atributos y métodos para leer, añadir, editar o eliminar información sobre los documentos que se cargan al sistema. También tiene métodos para obtener una lista de estaciones, glaciares, montañas, instituciones

y documentos, realizar búsquedas de información, generar gráficos temporales y descargar un archivo con los resultados de la búsqueda.

- `AsynController.cs`: Contiene atributos y métodos que permiten guardar la información que se encuentra dentro de cada documento. Posee los métodos para leer los documentos, validar que el documento tenga formato CSV, validar la estructura del archivo, guardar información sobre el documento, y almacenar la información de meteorología, glaciología, hidrología o metadatos en la base de datos. Las acciones se realizan de forma asincrónica.

Puesto que cada Controlador está asociado a una clase del Modelo, la Figura 2.7 presenta la relación que mantienen los Controladores con su respectiva Clase Entidad del Modelo. Las clases Entidad se representan en azul y los Controladores en Verde. Para la generación de Controladores se tomó en cuenta aquellas Clases Entidad que van a ser manipuladas de una u otra forma, por lo que no todas las Clases Entidad están asociadas a un Controlador.

En la Figura 2.8 se muestra el diagrama de clases de Controladores. Dado que los Controladores no se relacionan entre sí, sino que su relación es con su respectiva clase del Modelo, el diagrama no muestra las relaciones entre los mismos. Todos los Controladores heredan de la clase abstracta `Controller`, la misma que es propia de ASP.NET MVC, por ello se muestra dicha herencia en el diagrama.

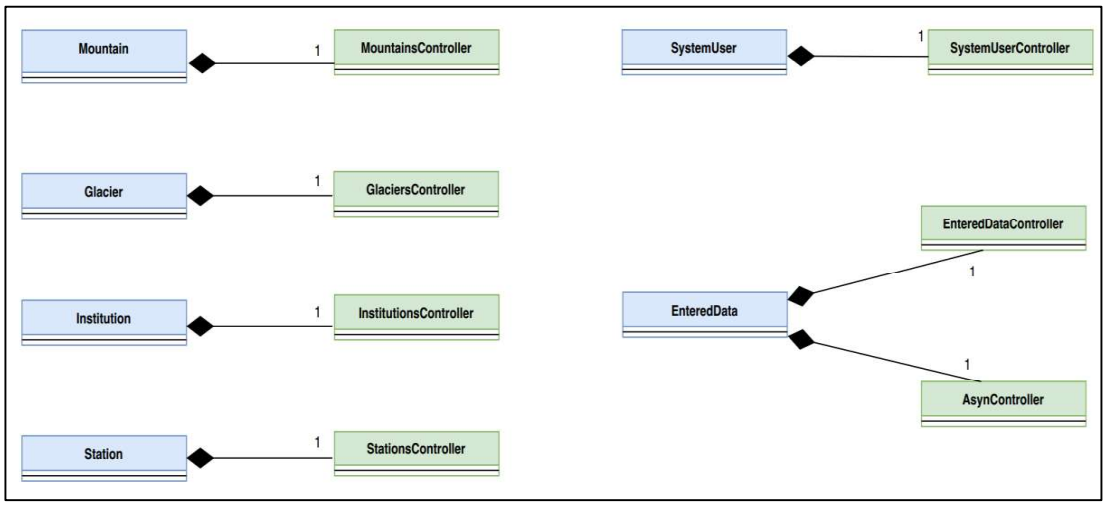


Figura 2.7 Relación entre los Controladores con su respectiva clase del Modelo

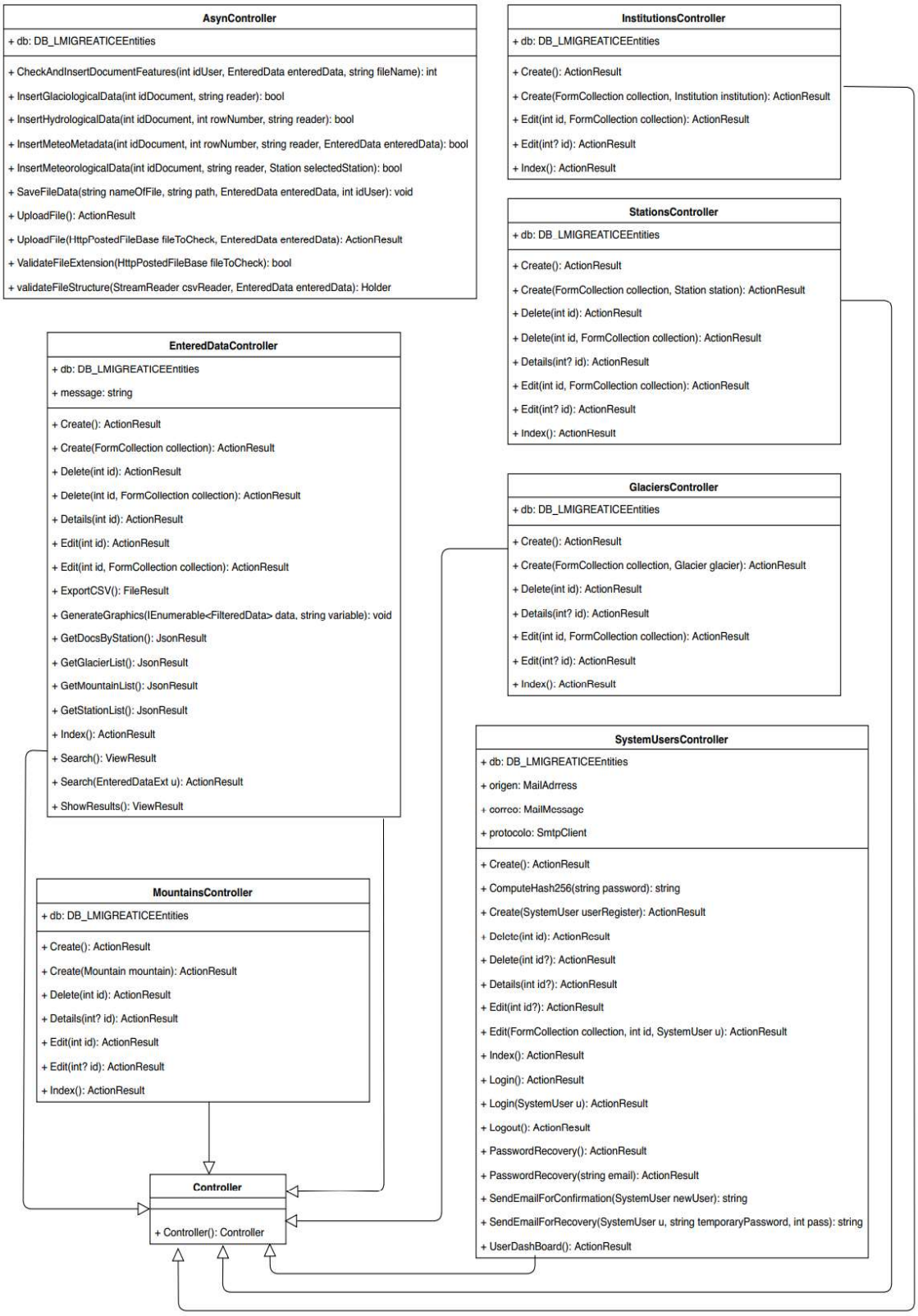


Figura 2.8 Diagrama de clases de Controladores

La Figura 2.9 muestra la relación existente entre los Controladores y la Clase Contexto.

Como se puede observar, existe una relación de 1 a 1 puesto que la Clase Contexto solo puede tener un Controlador con un nombre determinado y, cada Controlador solo puede estar asociada a una Clase Contexto dentro del Modelo.

La Clase Contexto es la que permitirá a los Controladores acceder a los datos del Modelo para realizar operaciones de inserción, modificación, eliminación o lectura en la base de datos.

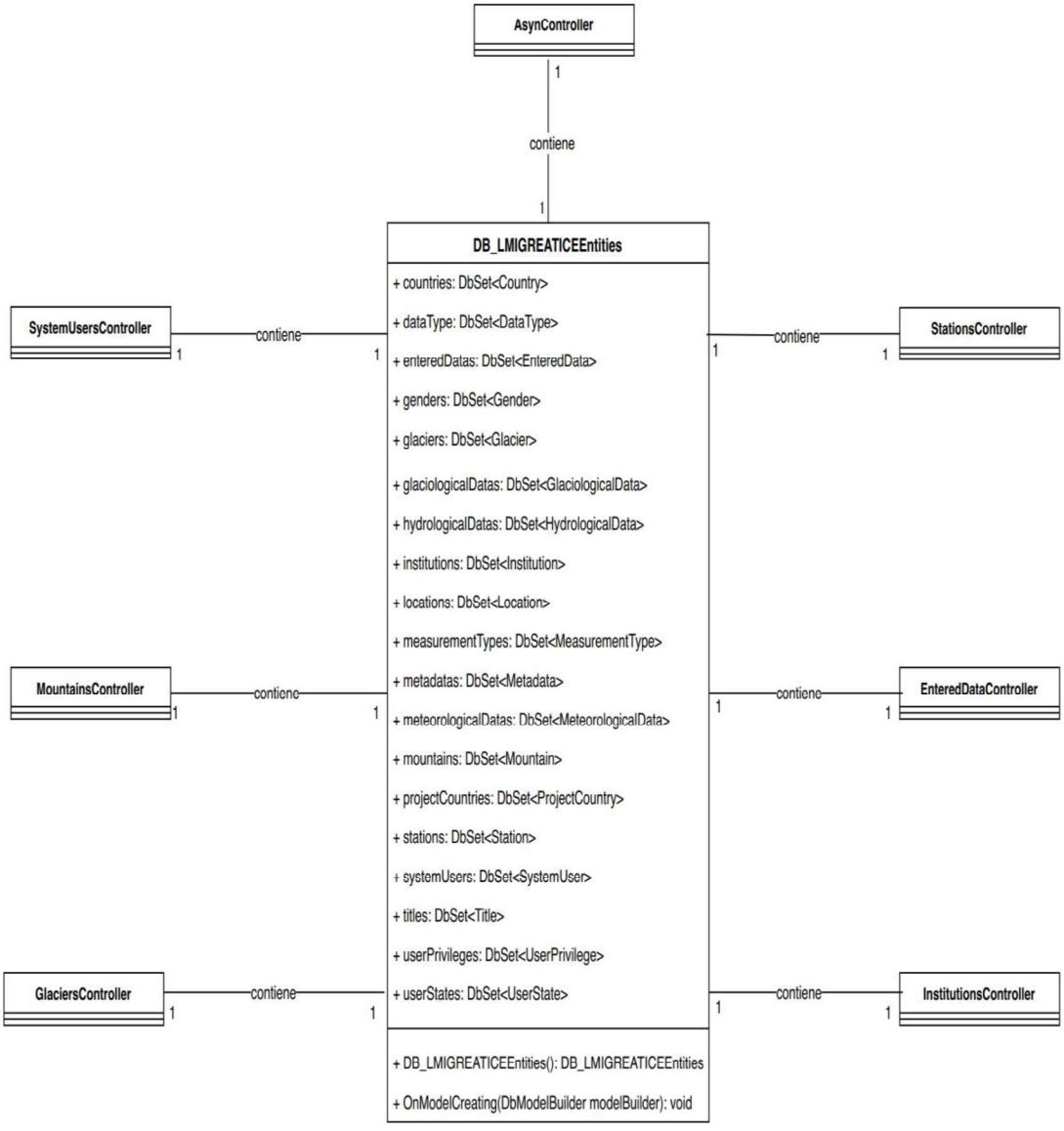


Figura 2.9 Relación entre los Controladores y la Clase Contexto

2.7 SKETCHES & WIREFRAMES

Los *sketches* se realizaron con base en los requerimientos solicitados, y mediante ellos se presentan las distintas interfaces de usuario que contendrá el sistema a desarrollar. A continuación, se muestran cada uno de los *sketches*.

En la Figura 2.10 se presenta la página principal de la aplicación web. En ella, se muestran varios elementos desplegables en los que se debe seleccionar los criterios de búsqueda para la información de los glaciares, como: país, variable, año, entre otros. También, se expone el mapa de países con los marcadores anclados a los lugares donde existen estaciones del proyecto LMI GREAT ICE. Además, existen dos opciones de acceso en la parte superior que redirigen hacia: registro de un nuevo usuario e inicio de sesión. Finalmente, en la parte superior de la pantalla se muestra un *banner* con una imagen.

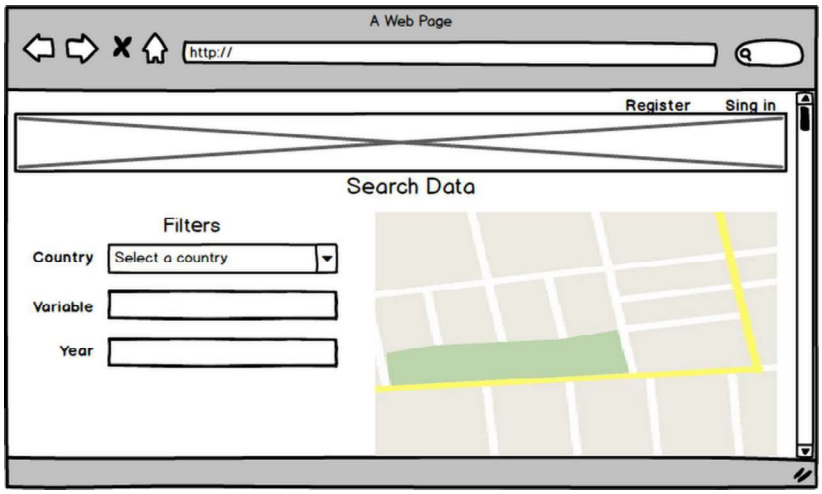


Figura 2.10 *Sketch* de la página principal

En la Figura 2.11 se muestra la página correspondiente a la creación de una nueva cuenta de usuario. Aquí se deben ingresar los datos personales del usuario como: nombre, apellido, correo electrónico, contraseña, género, título, fecha de nacimiento, país e institución. Dependiendo del tipo de dato a ingresar se presentan elementos como cuadros de texto, listas desplegables o calendario. Se mantiene el *banner* en la parte superior.

La Figura 2.12 corresponde a la pantalla de inicio de sesión. Para iniciar sesión, el usuario debe digitar su correo electrónico registrado y la respectiva contraseña. Es importante recalcar que, si la cuenta de usuario no está activada, no se podrá iniciar sesión.

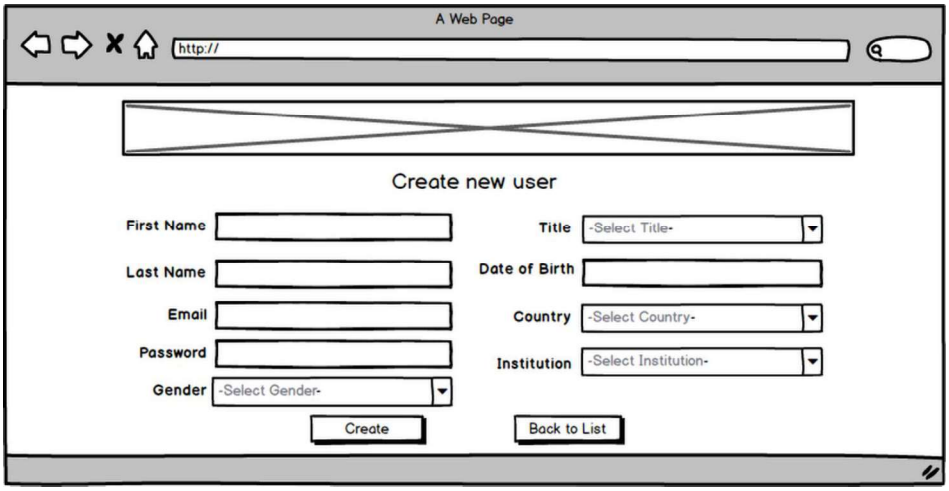


Figura 2.11 *Sketch* de la página para la creación de cuentas de usuario

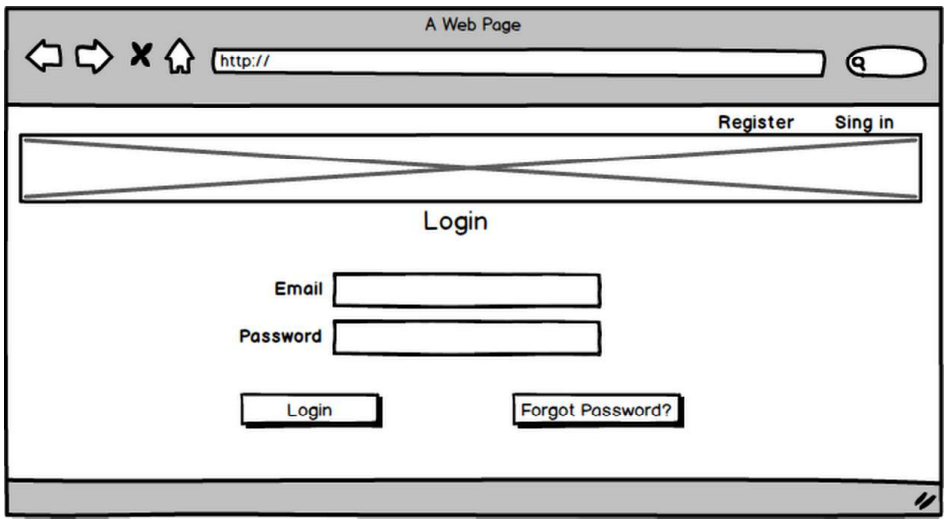


Figura 2.12 *Sketch* de la página de inicio de sesión

Si el usuario que inicia sesión tiene perfil `Administrator`, el sistema redirige a la página presentada en la Figura 2.13. En ella, se muestra por defecto una tabla con todos los usuarios registrados en el sistema, sus nombres, apellidos, correos electrónicos, países, instituciones a las que pertenecen, perfil de usuario asignado y estado de la cuenta de usuario. También se presentan opciones para crear un nuevo usuario, editar un usuario existente o eliminar un usuario registrado. Adicionalmente, en la parte izquierda se tiene un panel con varias opciones de acceso a otras partes del sistema: montañas, glaciares, estaciones, instituciones y carga de datos.

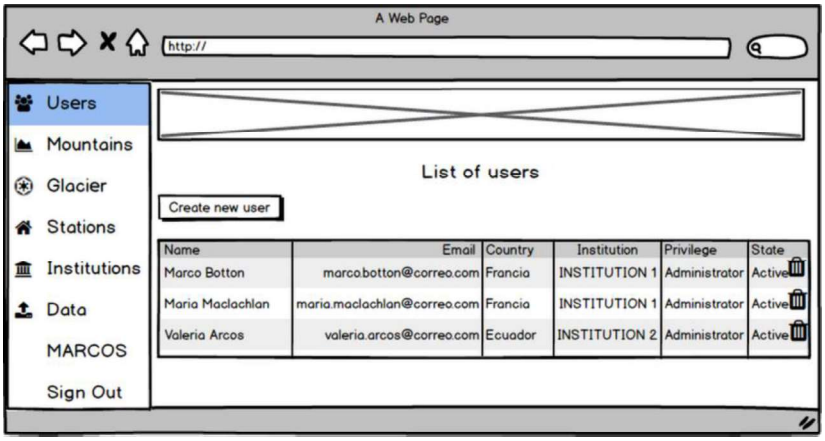


Figura 2.13 *Sketch* de la página principal al iniciar sesión con perfil `Administrator`

La Figura 2.14 exhibe la interfaz correspondiente a la edición de un usuario, por lo que se muestra toda la información del usuario a editar mediante elementos como cuadros de texto, listas desplegables o calendario. Dado que el usuario que inició sesión tiene perfil `Administrator`, el panel de la izquierda con acceso a otras opciones se sigue exponiendo.

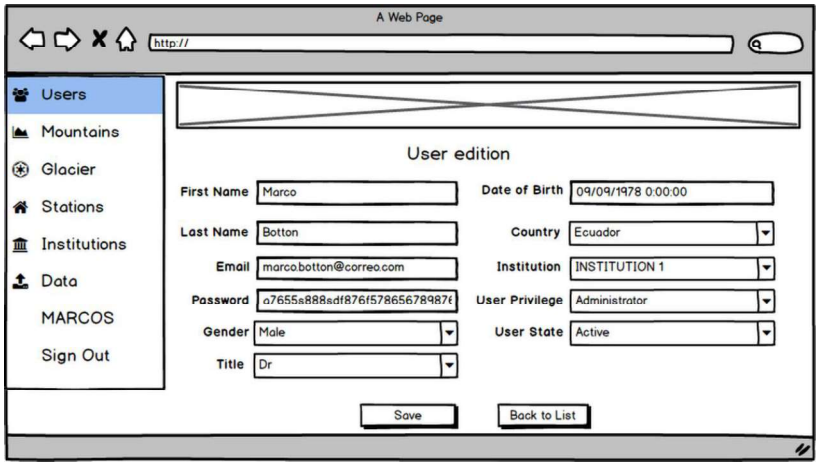


Figura 2.14 *Sketch* de la página para la edición de una cuenta de usuario

En caso de que el usuario que inicie sesión tenga perfil `Download` o `Download&Upload`, se muestra una pantalla como la presentada en la Figura 2.15. Dado que el perfil es restrictivo, únicamente se muestra la información del usuario que ha iniciado sesión, mas

no otros elementos. Desde aquí, el usuario que ha iniciado sesión también tiene la posibilidad de editar su información y eliminar su cuenta.

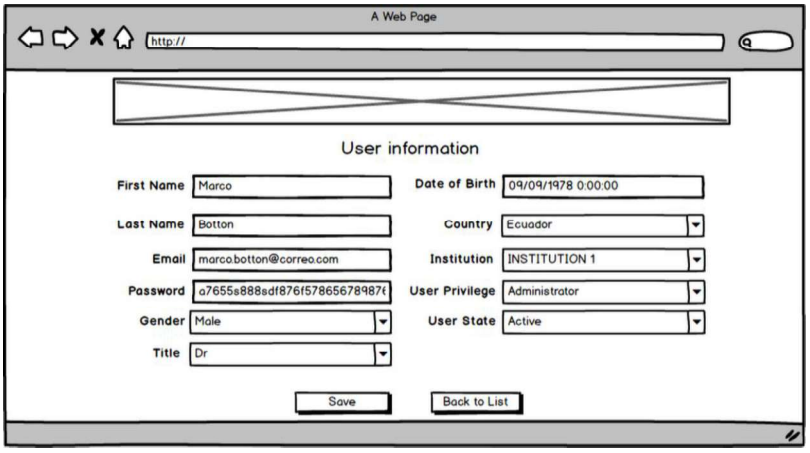


Figura 2.15 *Sketch* de la página de información de una cuenta de usuario

En la Figura 2.16 se expone la interfaz de usuario a la que se llega al seleccionar la opción `Stations` en el panel que está disponible únicamente para los usuarios con perfil `Administrator`. En la misma se muestra una tabla con las estaciones que recogen información actualmente para el proyecto LMI GREAT ICE. En la tabla constan el nombre de la estación, latitud, longitud, altitud, glaciar al que pertenece, ubicación en el glaciar, tipo de medida que recolecta y estado. También se muestran opciones para crear nuevas estaciones, editar la información de una estación específica o eliminar una estación.

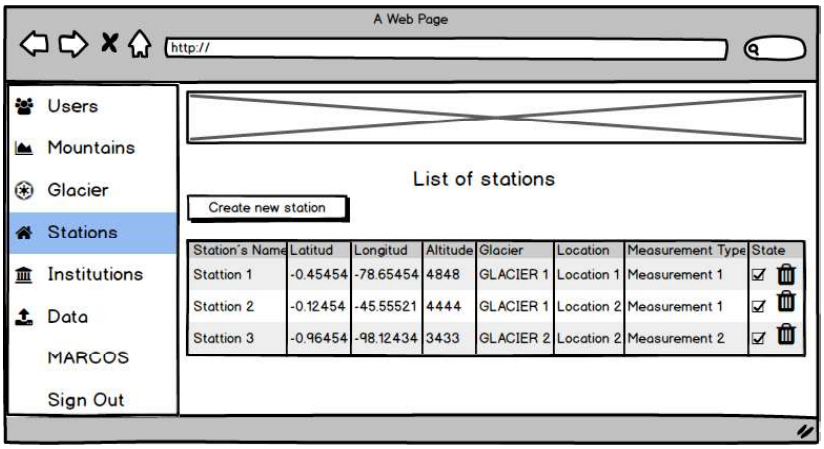


Figura 2.16 *Sketch* de la página para mostrar las estaciones

La Figura 2.17 presenta la interfaz de usuario correspondiente a la creación de una nueva estación. Aquí, el Administrador del sistema debe especificar toda la información requerida para añadir una estación, para ello, se tienen elementos como cuadros de texto y listas desplegables.

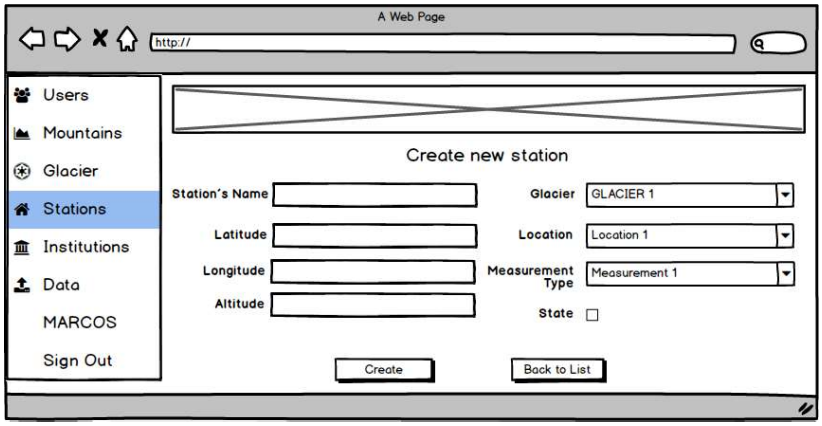


Figura 2.17 *Sketch* de la página para creación de nuevas estaciones

En caso de que el Administrador del sistema desee editar la información de una estación, se tendrá una interfaz como la presentada en la Figura 2.18.

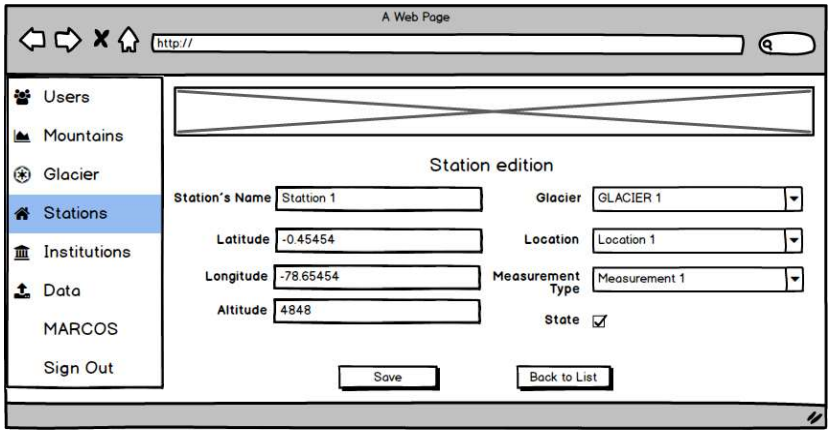


Figura 2.18 *Sketch* de la página para editar la información de una estación

Al seleccionar la opción `Data` en el panel, el sistema carga la interfaz de usuario que se exhibe en la Figura 2.19. Aquí se muestra una lista con los registros de todos los

documentos ingresados a la base de datos con información sobre glaciares. También se presentan opciones para añadir nuevos documentos, editar y eliminar los registros.

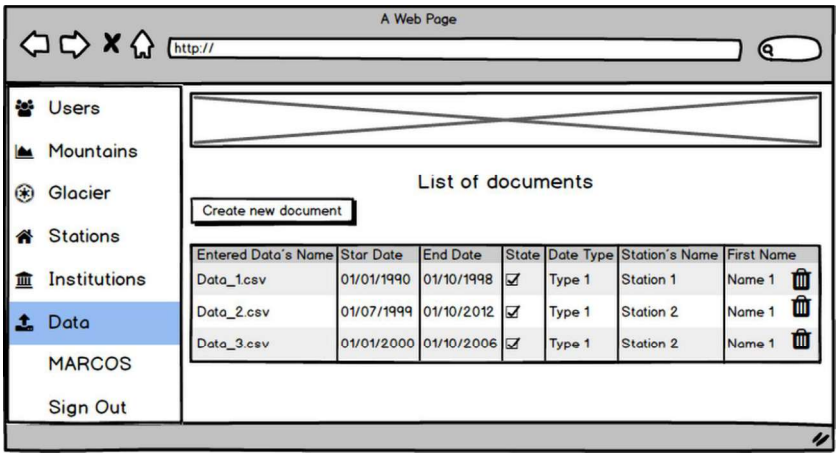


Figura 2.19 *Sketch* de la página para mostrar los documentos en el sistema

La Figura 2.20 presenta la interfaz correspondiente a cargar un nuevo documento con información de glaciares a la base de datos. Aquí es necesario indicar la fecha de inicio de la toma de datos, la fecha de finalización, el tipo de datos al que corresponde el documento (`Observation/Simulation`), el tipo de medida (`Meteorological, Glaciological o Hydrological`), la estación que recolectó esa información y seleccionar el documento a cargar.

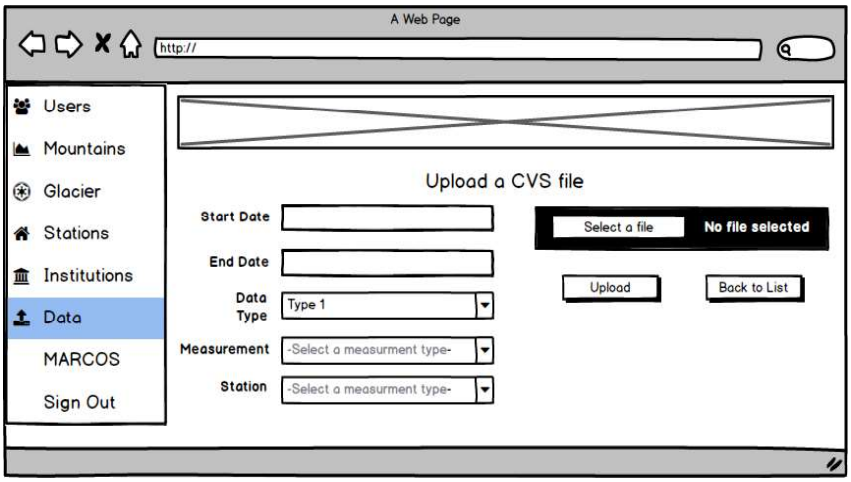


Figura 2.20 *Sketch* de la página para cargar nuevos documentos

En el panel al que tienen acceso únicamente los Administradores del sistema, se muestran otras opciones que se pueden seleccionar, como montañas (`Mountains`), glaciares (`Glaciers`) e instituciones (`Institutions`). Cada una de ellas permite añadir, editar o eliminar el elemento seleccionado y sus interfaces de usuario son similares a las presentadas.

A continuación, en la Figura 2.21 y la Figura 2.22, se presentan los principales *wireframes* elaborados para comprender el flujo entre Vistas.

Figura 2.21 *Wireframes* de la aplicación (parte 1 de 2)

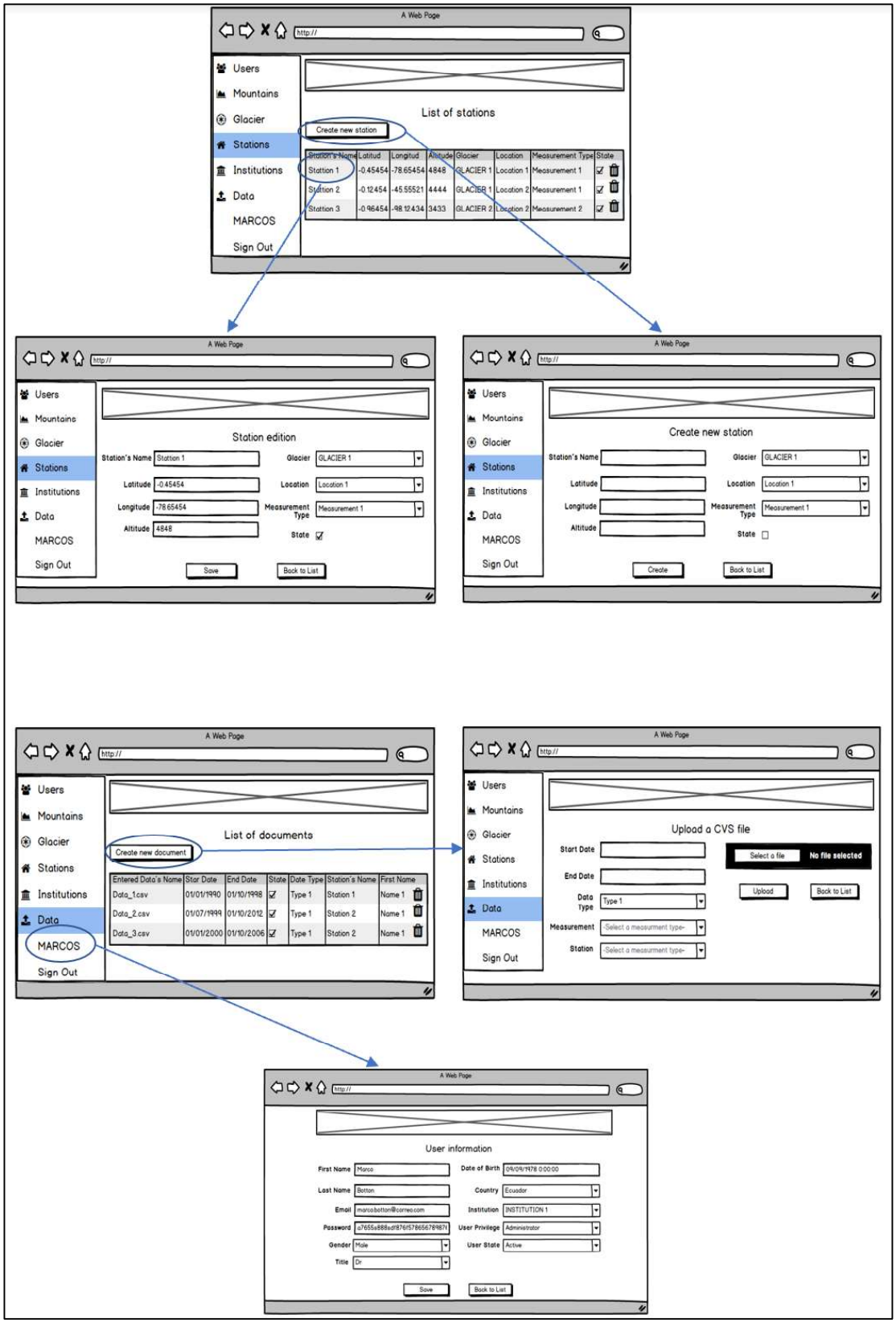


Figura 2.22 *Wireframes* de la aplicación (parte 2 de 2)

2.8 IMPLEMENTACIÓN

2.8.1 PROCESAMIENTO Y DEPURACIÓN DE INFORMACIÓN

Se realizó un procesamiento de los documentos que contenían información de los glaciares, provistos por miembros del proyecto LMI GREAT ICE. Se analizó la estructura de los documentos y los datos de los mismos. Se identificó que, en algunos documentos, ciertas celdas tenían el valor `#VALEUR!` por un error en la fórmula de cálculo de las variables. En estos casos exclusivamente, el valor mencionado fue reemplazado por `-6999`, que indica que no existe registro de medición en dicha celda. Esto se hizo con el objetivo de que no existan errores al momento de almacenar la información en la base de datos, debido a que se realiza una validación de los tipos de datos en función de las variables medidas. Este cambio de valor se esquematiza en la Figura 2.23.

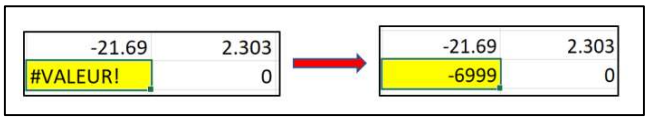


Figura 2.23 Cambio de valor en celdas de documentos

Adicionalmente, se definieron plantillas que deben cumplir los documentos para cargarlos al sistema en función del tipo de medida (`Glaciological`, `Meteorological` o `Hydrological`). Las plantillas establecidas se encuentran en el ANEXO B.

2.8.2 BASE DE DATOS

Para la implementación de la base de datos se utilizó SQL Server Management Studio 2017. Se realizó la codificación de *scripts* en lenguaje SQL. Concretamente se utilizaron tres *scripts*: uno que contiene el código para la creación de la base de datos y las tablas requeridas, otro para poblar ciertas tablas con valores por defecto, y un *script* que permitió crear y cargar la tabla `country` específicamente. Este último fue utilizado tomando como referencia la información provista en [46]. Los *scripts* empleados se encuentran en el ANEXO C. A continuación, se presentan partes del código utilizado. En el Código 2.1 se indica la creación de la base de datos `DB_LMIGREATICE` (línea 1), y la utilización de la misma para generar las tablas (línea 2).

```
1.  create database DB_LMIGREATICE
2.  use DB_LMIGREATICE
```

Código 2.1 Creación de la base de datos `DB_LMIGREATICE`

En el Código 2.2 se muestra un ejemplo del código necesario para la creación de una tabla con sus distintos atributos. Se puede observar la sentencia de creación de la tabla en la línea 109, en este caso se trata de la tabla `meteorologicalData`. En la línea 110 se declara el atributo que actuará como clave primaria de la tabla, cuyo tipo de dato es entero y se incrementa automáticamente de 1 en 1. Desde la línea 111 a la línea 130 se definen los atributos de la tabla, estableciendo el tipo de dato para cada uno. Puede observarse que existen algunos atributos que son obligatorios y otros que no lo son. Finalmente, en la línea 131 se establece una llave foránea para relacionar esta tabla con la tabla `enteredData`.

```
109.    create table meteorologicalData(
110.    idMeteorologicalData int identity(1,1) primary key,
111.    dateData dateTime not null,
112.    inShortWaveRadiation float not null,
113.    outShortWaveRadiation float not null,
114.    albedo float not null,
115.    inLongWaveRadiation float,
116.    outLongWaveRadiation float,
117.    relativeHumidity float not null,
118.    ventilatedAirTemperature float not null,
119.    nonVentilatedAirTemperature float,
120.    windSpeed float not null,
121.    windDirection float not null,
122.    precipitationAmount float,
123.    precipitationRate float,
124.    distanceSensor_Snow float,
125.    snowDepth float,
126.    groundFlux float,
127.    groundTemp_3cm float,
128.    groundTemp_10cm float,
129.    groundTemp_30cm float,
130.    idEnteredData int not null,
131.    foreign key (idEnteredData) references enteredData(idEnteredData)
132.    );
```

Código 2.2 Creación de la tabla `meteorogicalData`

En el Código 2.3 se muestra una parte del *script* utilizado para poblar la base de datos. En la línea 1 se indica la utilización de la base de datos `DB_LMIGREATICE`. En la línea 2 se indica en qué tabla se van a cargar los datos y en qué columna específicamente, en este caso, los datos se insertarán en la tabla `institution`, en la columna

`nameInstitution`. En la línea 3 se especifican los valores a cargar en la columna indicada.

```
1. use DB_LMIGREATICE

2. insert into institution(nameInstitution)
3. values ('LMI GREATICE'),('ESCUELA POLITÉCNICA NACIONAL')
```

Código 2.3 Inserción de valores en la tabla `institution`

2.8.3 APLICACIÓN WEB

La aplicación web fue codificada haciendo uso del IDE Visual Studio 2017. El lenguaje de programación utilizado fue C#.

Se creó un proyecto, `WebApp_LMIGREATICE`, dentro de la solución del mismo nombre. Este proyecto fue creado utilizando el *framework* .NET 4.6.1. La plantilla de creación del proyecto se muestra en la Figura 2.24.

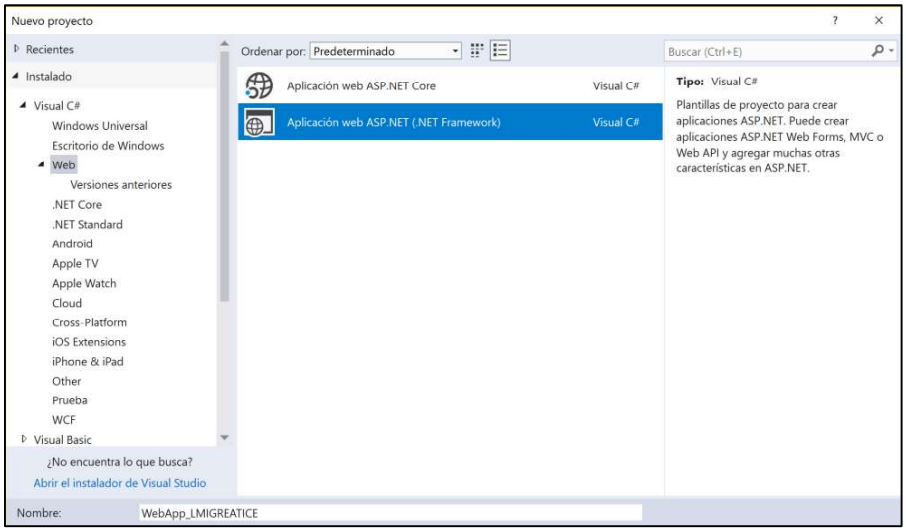


Figura 2.24 Creación del proyecto `WebApp_LMIGREATICE`

La aplicación web fue implementada en ASP.NET MVC, un *framework* que viene por defecto en el IDE Visual Studio 2017. En la Figura 2.25 se muestra la elección de la plantilla MVC en el proyecto creado.

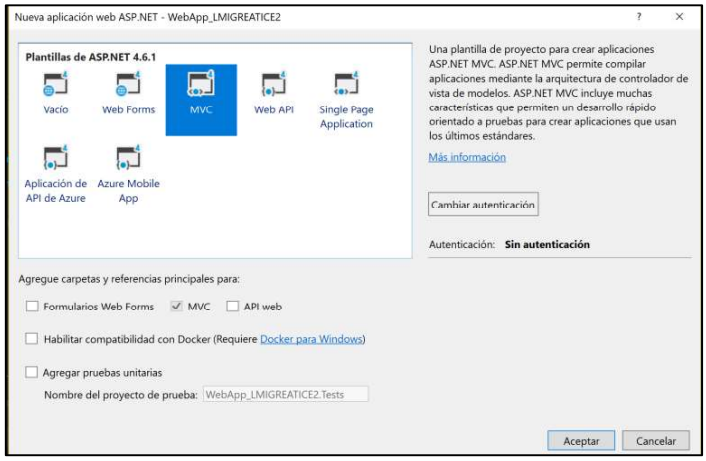


Figura 2.25 Elección de plantilla MVC en el proyecto creado

El proyecto contiene, entre otras cosas, las clases del Modelo; los Controladores utilizados; las Vistas creadas y el archivo `Web.config` en el que consta la cadena de conexión hacia la base de datos. El código del proyecto se encuentra en el ANEXO D.

2.8.3.1 Modelo

El Modelo que se genera mediante Entity Framework se denomina EDM (*Entity Data Model*). Para la generación del EDM a partir de la base de datos mediante Entity Framework, debe agregarse un nuevo elemento en la carpeta `Models` de tipo ADO.NET *Entity Data Model*, tal como se muestra en la Figura 2.26. En la Figura 2.27 se presenta la plantilla a escoger para generar el Modelo considerando el enfoque *DataBase First*. Al escoger esta plantilla, se genera un archivo con extensión `.edmx`.

Al crearse el Modelo `LmiGreatIceModel`, se crearon las siguientes Clases Entidad dentro del mismo: `country`, `dataType`, `enteredData`, `gender`, `glacier`, `glaciologicalData`, `hydrologicalData`, `institution`, `location`, `measurementType`, `metadata`, `meteorologicalData`, `mountain`, `projectCountry`, `station`, `systemUser`, `title`, `userPrivilege` y `userState`. Cada clase Entidad representa a una tabla en la base de datos.

Los atributos de las Clases Entidad fueron complementados con *Data Annotations* para proveer validaciones en el lado del cliente, sobre los datos ingresados por el usuario. El Código 2.4 muestra un ejemplo de clase Entidad creada en el Modelo, y la utilización de *Data Annotations*. La clase `glacier` contiene un constructor (líneas 45 a 48); también se

definen los atributos de la clase con sus respectivos tipos de datos (líneas 50 a 65). Se puede observar que en ciertos atributos se colocan *Data Annotations*, por ejemplo, para el atributo `nameGlacier` se indica que es un atributo que el usuario debe llenar obligatoriamente, caso contrario se mostrará un mensaje de error (línea 52), además, el nombre a mostrar en la interfaz de usuario será "`Name Glacier`" (línea 53). Los atributos contenidos en las líneas 50 a 62 representan propiedades escalares, cada uno se asocia con una única columna en la tabla correspondiente de la base de datos. El atributo `mountain` (línea 64) es una propiedad de navegación de referencia que representa una relación con otra entidad, con una multiplicidad de uno (1). Finalmente, el atributo `station` (línea 65) es una propiedad de navegación de colección que representa una relación con otra entidad, con una multiplicidad de varios (*).

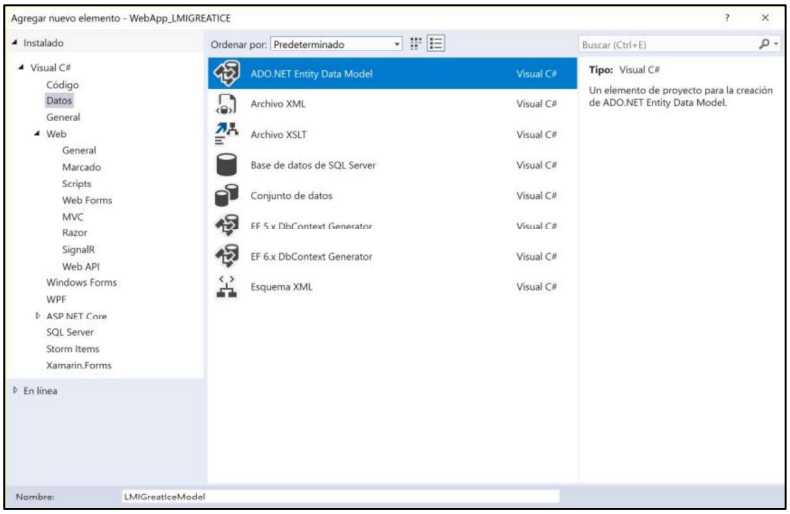


Figura 2.26 Añadir un nuevo elemento para generar el Modelo

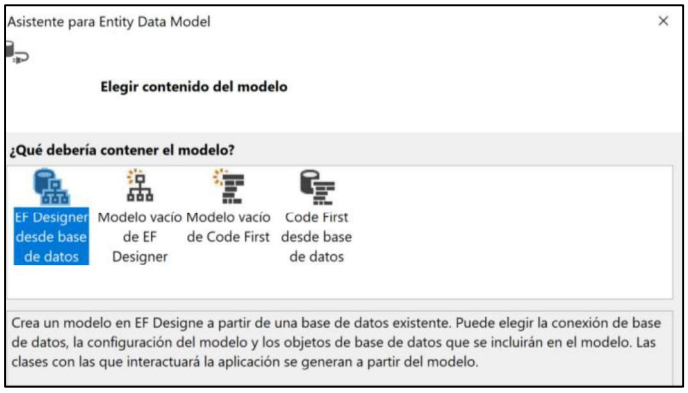


Figura 2.27 Generación del Modelo a partir de la base de datos

```
43  public partial class glacier
44  {
        0 referencias | 0 excepciones
45      public glacier()
46      {
47
48      }
49
        2 referencias | 0 excepciones
50      public int idGlacier { get; set; }
51
52      [Required(ErrorMessage = "Field Required")]
53      [DisplayName("Name Glacier")]
        2 referencias | 0 excepciones
54      public string nameGlacier { get; set; }
55
56      [Required(ErrorMessage = "Field Required")]
57      [DisplayName("Mountain")]
        2 referencias | 0 excepciones
58      public int idMountain { get; set; }
59
60      [Required(ErrorMessage = "Field Required")]
61      [DisplayName("State")]
        1 referencia | 0 excepciones
62      public bool stateR { get; set; }
63
        0 referencias | 0 excepciones
64      public virtual mountain mountain { get; set; }
        0 referencias | 0 excepciones
65      public virtual ICollection<station> stations { get; set; }
66  }
```

Código 2.4 Ejemplo de clase Entidad del Modelo

También se creó la Clase Contexto `DB_LMIGREATICEEntities1`, esta clase permitirá establecer una sesión con la base de datos para guardar información en la misma o retornarla desde ella. El Código 2.5 muestra un ejemplo de utilización de la Clase Contexto y de la clase Entidad `glacier`. En la línea 13 se crea una instancia de la Clase Contexto `DB_LMIGREATICEEntities1`. Desde la línea 16 a la 20 se define un método que lee todos los registros de la tabla `glaciers` y los asigna a una variable para posteriormente desplegar en una Vista la lista retornada. Es necesario hacer uso de la instancia de la Clase Contexto, creada anteriormente, para acceder a la información de la tabla `glaciers` en la base de datos (línea 18).

```
13      DB_LMIGREATICEEntities1 db = new DB_LMIGREATICEEntities1();
14
15      // GET: glaciers
        0 referencias | 0 solicitudes | 0 excepciones
16      public ActionResult Index()
17      {
18          var glaciers = db.glaciers.ToList();
19          return View(glaciers);
20      }
```

Código 2.5 Método para listar las entidades del Modelo

Adicionalmente, se crearon de forma manual algunas clases extra con distintos propósitos:

- La clase `FilteredData` permite tener un resumen con la información extraída de los documentos al realizar una búsqueda. De esta manera, al buscar información sobre una variable, la clase permite asociar la fecha y hora con los valores medidos de dicha variable. Esto facilita la generación de gráficas temporales.

  En el Código 2.6 se muestra un ejemplo de la utilización de esta clase. En la línea 488 se crea una lista de objetos de tipo `FilteredData` . Desde la línea 489 a la línea 500 se itera a través de una lista que contiene múltiples filas y columnas con información de los documentos. Se verifica que los valores en la lista sean válidos. Luego se crea un objeto de la clase `FilteredData` al que se le asigna en cada iteración, únicamente los valores de las dos primeras columnas de la lista (`key1` y `key2`) que corresponden a fecha y valor, respectivamente. Por último, se añade en cada iteración el objeto creado a la lista de objetos `FilteredData` que se creó al inicio.

```
488             List<FilteredData> summary = new List<FilteredData>();
489             foreach (var item in searchedData)
490             {
491                 if (Double.Parse(item.Get<double>(variable).ToString()) != -6999)
492                 {
493                     FilteredData aux = new FilteredData()
494                     {
495                         key1 = item.dateData,
496                         Key2 = Double.Parse(item.Get<double>(variable).ToString())
497                     };
498                     summary.Add(aux);
499                 }
500             }
```

Código 2.6 Ejemplo de uso de la clase `FilteredData`

- La clase `Holder` por su parte, fue creada para llevar un control de las validaciones y de los mensajes a mostrar en cada caso. Un ejemplo de uso de la clase `Holder` se muestra en el Código 2.7. Se puede observar que se crea un objeto de esta clase y se le asigna el resultado de la llamada a un método para validar la estructura de un archivo (línea 167). Este objeto contiene un atributo `status` de tipo `boolean` y un atributo `message` de tipo `string`.

  Si es que el valor del atributo `status` es `True`, se hace una llamada al método para guardar información (líneas 168 a 171). Caso contrario se despliega un mensaje con el error contenido en el atributo `message` y se redirige hacia otra acción del controlador (líneas 172 a 176).

```
167     Holder hold = validateFileStructure(csvReader, enteredData); //validación de estructura de archivo
168     if (hold.status == true)
169     {
170         saveFileData(csvReader, fileToCheck, enteredData);
171     }
172     else
173     {
174         Session["error"] = hold.message;
175         return RedirectToAction("UploadFile");
176     }
```

Código 2.7 Ejemplo de uso de la clase `Holder`

- La clase `StructureValidation` se creó con el objetivo de contener los métodos necesarios para validar la estructura interna de cada uno de los documentos al momento de subirlos a la base de datos.

  Los métodos validan distintos aspectos en función del tipo de documento a cargar (`Meteorological, Glaciological o Hydrological`).

  Por ejemplo, se validan tipos de datos, nombres de columnas, valores, entre otros. Un ejemplo de uso de esta clase se muestra en el Código 2.8. Se puede observar que en la línea 274 se realiza un condicional para verificar que la estación seleccionada corresponda al tipo de medida Meteorología (`idMeasurementType = 1`). Si es así, se llama al método `checkMeteorologicalStructure` que se encuentra en la clase `StructureValidation`, para validar la estructura de un documento de Meteorología pasándole los parámetros necesarios, y se le asigna el resultado de la validación al objeto `hold` (línea 276).

```
274             if (selectedStation.idMeasurementType == 1) //VALIDAR ESTRUCTURA DATOS METEOROLOGICOS
275             {
276                 hold = new StructureValidation().checkMeteorologicalStructure(csvReader, enteredData, selectedStation);
277             }
```

Código 2.8 Ejemplo de utilización de la clase `StructureValidation`

En el Código 2.9 se muestra el método `ValidateNumbers`, una porción de la clase `StructureValidation` que permite validar si es que los datos pasados como parámetro de entrada son números o no. Esto con el objetivo de que, al momento de cargar el documento, no se generen errores por el tipo de dato.

El método `ValidateNumbers` es a su vez utilizado por otros métodos de la misma clase.

```
36 referencias | 0 excepciones
public bool validateNumbers(string forValidate) //Valida si los valores pasados son números
{
    bool isNumber = false;
    double test;
    try
    {
        if(double.TryParse(forValidate, out test))
        {
            isNumber = true;
        }
    }
    catch
    {
        isNumber = false;
    }
    return isNumber;
}
```

Código 2.9 Método `ValidateNumbers`

En el Código 2.10 se muestra una parte del método `checkMeteorologicalStructure`. La porción de código que se puede observar se encarga de verificar que el valor de una celda en particular no sea nulo y que dicho valor sea un número, puesto que la columna a la que pertenece esa celda únicamente debe contener números.

Se generan también mensajes para proveer al usuario de una retroalimentación que le permita observar con claridad si existió algún error y cuál es el error.

```
//VALIDACION AUX[1]
if (aux[1] != null && validateNumbers(aux[1]))
{
    auxStatus = (auxStatus && true);
}
else
{
    auxStatus = (auxStatus && false);
    error += "\n ROW" + " " + rowNumber + ":" + "VALUES ON SECOND COLUMN (Swinc) MUST BE NUMBERS";
}
```

Código 2.10 Parte del método `checkMeteorologicalStructure`

- La clase `EnteredDataExt` hereda de `enteredData`, y contiene los atributos propios `year` y `variable`. Esta clase permite pasar los datos ingresados por el usuario en la Vista de búsqueda al método que se encarga de la búsqueda de información.

  El uso de la clase `EnteredDataExt` se muestra en el Código 2.11. Aquí se observa que en el método `Search` se recibe como parámetro de entrada un objeto de la clase

`EnteredDataExt` (línea 480), mismo que contiene todos los datos especificados por el usuario en la Vista respectiva para realizar una búsqueda de información.

```
479         [HttpPost]
            0 referencias | 0 solicitudes | 0 excepciones
480         public ActionResult Search(EnteredDataExt u)
481         {
482             int year = u.year;//Convert.ToDateTime("2007-01-01");
483             string variable = u.variable;
```

Código 2.11 Ejemplo de utilización de la clase `EnteredDataExt`

- La clase `EnteredDataSimplify` contiene los atributos `idDoc`, `idStation`, `name`, `startDate` y `endDate`. Se utiliza al obtener y desplegar una lista de documentos cuando se selecciona un marcador del mapa interactivo, con base en la estación asociada a dicho marcador.

  El Código 2.12 muestra un ejemplo del uso de esta clase. Lo que se hace desde la línea 133 a la 140, es obtener una lista de registros de la tabla `enteredDatas` donde el valor del atributo `idStation` coincide con el que se ha indicado. Luego, de esta lista de registros se selecciona únicamente las columnas que se necesitan, para ello se declara un objeto anónimo de la clase `EnteredDataSimplify` y se lo inicializa con los valores requeridos. Finalmente se lista la información obtenida.

```
133         var docs = db.enteredDatas.Where(x => x.idStation == idStation).Select(p => new EnteredDataSimplify
134         {
135             idDoc = p.idEnteredData,
136             idStation = p.idStation,
137             name = p.nameEnteredData,
138             startDate = p.startDate.ToString(),
139             endDate = p.endDate.ToString()
140         }).ToList();
```

Código 2.12 Ejemplo de utilización de la clase `enteredDataSimplify`

- La clase `StationTrick` se utiliza para devolver una lista de todas las estaciones guardadas en la base de datos y cuyo estado es `Active`. Su uso se muestra en el código Código 2.13. La porción de código presentada se encarga de obtener una lista

de registros de la tabla `stations`, donde el valor del atributo `stateR` es igual a `True`. Después se seleccionan únicamente las columnas de interés en la lista de registros. Para ello se declara un objeto anónimo de la clase `StationTrick` y se lo inicializa con los valores de las columnas que se desean. Finalmente se lista la información y se la guarda en un contenedor `ViewBag` para su uso posterior.

```
462         ViewBag.fullStation = db.stations.Where(i => i.stateR == true).Select(p => new StationTrick
463         {
464             idStation = p.idStation,
465             nameStation = p.nameStation,
466             idLocation = p.idLocation,
467             latitudeStation = p.latitudeStation,
468             longitudeStation = p.longitudeStation,
469             altitudeStation = p.altitudeStation,
470             idGlacier = p.idGlacier,
471             idMeasurementType = p.idMeasurementType
472         }).ToList();
473         return View();
474     }
```

Código 2.13 Ejemplo de utilización de la clase `StationTrick`

2.8.3.2 Controladores

Una vez generado el modelo, se crearon los Controladores respectivos en función de las necesidades. Para la codificación de los Controladores, se utilizó *Scaffolding,* con lo cual se generaron plantillas genéricas que aceleraron el desarrollo.

Los Controladores permiten interactuar con el Modelo y las Vistas. Además, contienen métodos que permiten realizar acciones para responder a las solicitudes del usuario.

La primera sentencia dentro de un Controlador debe ser la instanciación de la Clase Contexto. Como anteriormente se explicó, la Clase Contexto permite establecer una sesión con la base de datos para acceder o manipular la información que contiene.

Por lo tanto, al crear un objeto de la Clase Contexto, se tiene acceso a la base de datos y a su información.

En el Código 2.14 se indica la sentencia para crear un objeto de la Clase Contexto.

```
13          DB_LMIGREATICEEntities1 db = new DB_LMIGREATICEEntities1();
```

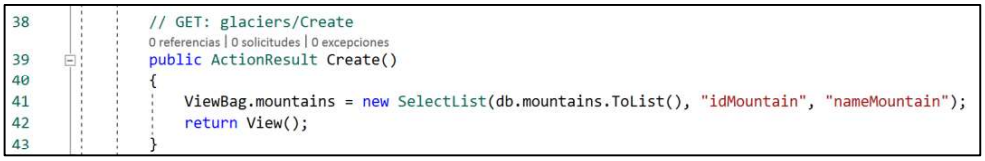
Código 2.14 Instanciación de la Clase Contexto

Los Controladores creados son:

- `glaciersController`: Contiene métodos de acción necesarios para leer y manipular la información de la tabla `glacier`. Los métodos incluyen creación, actualización y eliminación de registros de glaciares.
- `mountainsController`: Contiene métodos de acción necesarios para leer y manipular la información de la tabla `mountain`. Los métodos incluyen creación, actualización y eliminación de registros de montañas.
- `institutionsController`: Contiene métodos de acción necesarios para leer y manipular la información de la tabla `institution`. Los métodos incluyen creación y actualización de registros de instituciones.
- `stationsController`: Contiene métodos de acción necesarios para leer y manipular la información de la tabla `station`. Los métodos incluyen creación, actualización y eliminación de registros de estaciones.
- `systemUsersController`: Contiene métodos de acción necesarios para leer y manipular la información de la tabla `systemUser`. Los métodos incluyen creación, actualización y eliminación de registros; inicio de sesión; cierre de sesión; cifrado de contraseña; recuperación de contraseña y envío de correo de confirmación.
- `enteredDataController`: Contiene métodos de acción para leer y manipular la información de la tabla `enteredData`. Los métodos incluyen creación y eliminación de registros; obtención de una lista de los glaciares, montañas, estaciones, instituciones y documentos almacenados en la base de datos; búsqueda de información; generación de gráficos temporales y descarga de documentos.
- `AsynController`: Contiene métodos de acción que permiten guardar la información que se encuentra dentro de cada documento. Posee los métodos para leer los documentos; validar que la extensión del archivo corresponda a un documento CSV; validar la estructura del archivo; guardar información sobre el documento; y almacenar la información de meteorología, glaciología, hidrología o metadatos en la base de datos. Las acciones se realizan de forma asincrónica.

Los métodos de acción para creación, actualización y eliminación de registros son similares en todos los Controladores, por lo que se explicará un ejemplo de cada acción únicamente. En el Código 2.15 se explica el método `Create` implementado con `GET`, perteneciente al

Controlador `glaciersController`. Este método se encarga de mostrar los elementos e información necesaria en la Vista respectiva en el momento en el que un usuario desea crear un nuevo registro de tipo `glacier`. Como se puede observar, el tipo de acción del método es `ActionResult` (línea 39). En la línea 41 se hace uso del contenedor de datos `ViewBag`. En este contenedor se cargan todos los registros de montañas (`mountain`) que contiene la base de datos para desplegarlas al usuario, puesto que, para crear un nuevo registro de tipo `glacier`, es necesario seleccionar la montaña (`mountain`) a la que pertenece. Finalmente, en la línea 42 se especifica que el método retorna una Vista.



```
38      // GET: glaciers/Create
        0 referencias | 0 solicitudes | 0 excepciones
39      public ActionResult Create()
40      {
41          ViewBag.mountains = new SelectList(db.mountains.ToList(), "idMountain", "nameMountain");
42          return View();
43      }
```

Código 2.15 Método `Create` implementado con `GET`

En el Código 2.16 se explica el método `Create` implementado con `POST`, perteneciente al Controlador `glaciersController`. Este método se encarga de añadir un nuevo registro de tipo `glacier` en la base de datos. Lo primero que se puede observar es que este método lleva un atributo [`HttpPost`], antes de su definición, para especificar que está implementado con `POST` (línea 54). Esto permite diferenciarlo del método `Create` implementado con `GET`. En la línea 55 se muestra que el tipo de acción del método es `ActionResult`, y que recibe como parámetros de entrada los datos ingresados por el usuario en la Vista de creación respectiva.

Se realiza una consulta a la tabla `glaciers` de la base de datos buscando un registro cuyo atributo `nameGlacier` sea igual al indicado por el usuario, y se lo almacena en la variable `newGlacier`. Si la consulta retorna un valor nulo, implica que en la base de datos no existen registros de glaciares con el nombre especificado, por lo que se procede a agregar el nuevo registro y guardar los cambios realizados (líneas 59 a 67). Para agregar el nuevo registro a la tabla `glaciers`, se hace uso de la instancia de la Clase Contexto, `db`. Lo mismo sucede para guardar los cambios generados en la base de datos. En la línea 67 se indica que el método redireccionará hacia la acción `Index` del mismo Controlador. También se añade captura de excepciones a través de bloques `try{}` y `catch{}`.

```
54      [HttpPost]
        0 referencias | 0 solicitudes | 0 excepciones
55      public ActionResult Create(FormCollection collection, glacier glacier)
56      {
57          try
58          {
59              var newGlacier = db.glaciers.Where(a => a.nameGlacier.Equals(glacier.nameGlacier)).FirstOrDefault();
60              if (newGlacier == null)
61              {
62                  db.glaciers.Add(glacier);
63                  db.SaveChanges();
64                  Session.Add("message", "Glacier added successfully");
65                  Session["error"] = null;
66                  Session["notification"] = null;
67                  return RedirectToAction("Index");
68              }
69              else
70              {
71                  Session.Add("error", "This glacier already exists!");
72                  Session["message"] = null;
73                  Session["notification"] = null;
74                  return RedirectToAction("Create");
75              }
76          }
77          catch
78          {
79              return View();
80          }
81      }
```

Código 2.16 Método `Create` implementado con `POST`

En el Código 2.17 se explica el método `Edit` implementado con `GET`, perteneciente al Controlador `glaciersController`. En la línea 77 se puede observar que el método recibe como parámetro de entrada el `id` del registro de glaciar que se desea editar. En la línea 79 se hace uso de una expresión lambda para consultar en la tabla `glacier` de la base de datos, aquellos registros de glaciar cuyo `id` coincida con el parámetro de entrada indicado. El objeto retornado se asigna a una variable de tipo `glacier`. En las líneas 80 a 83, se realiza una validación para verificar si el objeto es nulo, en caso de serlo se retorna un error de tipo `HttpNotFound`. En la línea 84 se realiza una consulta a la tabla `mountains` de la base de datos, se retorna una lista de sus registros y se los almacena en el contenedor `ViewBag` indicado para usarlos posteriormente. Finalmente, en la línea 85 se retorna una Vista con el registro del glaciar a editar. En las consultas a la base de datos se utiliza la instancia de la Clase Contexto, `db`.

En el Código 2.18 se explica el método `Edit` implementado con `POST`, perteneciente al Controlador `glaciersController`. La línea 89 especifica el atributo [`HttpPost`] para indicar que este método se implementa con `POST`. El método recibe como parámetro de entrada el `id` del registro de glaciar que se desea editar (línea 90). Se realiza una consulta a la tabla `glacier` de la base de datos, para buscar el registro de dicha tabla cuyo `id` coincida con el parámetro de entrada. El objeto retornado se asigna a la variable `glacier` (línea 95). Se actualiza el Modelo enviando el nuevo objeto `glacier` que ha sido

modificado por el usuario (línea 96). Se hace uso de la instancia de la Clase Contexto para guardar los cambios que se hicieron (línea 97), y se redirige a otra acción para desplegar la Vista requerida (línea 98). Todo esto dentro de bloques `try{}` y `catch{}` para captura de excepciones.

```
76      // GET: glaciers/Edit/5
        0 referencias | 0 solicitudes | 0 excepciones
77      public ActionResult Edit(int id)
78      {
79          var glacier = db.glaciers.Single(i => i.idGlacier == id);
80          if (glacier == null)
81          {
82              return HttpNotFound();
83          }
84          ViewBag.mountains = new SelectList(db.mountains.ToList(), "idMountain", "nameMountain");
85          return View(glacier);
86      }
```

Código 2.17 Método `Edit` implementado con `GET`

```
88      // POST: glaciers/Edit/5
89      [HttpPost]
        0 referencias | 0 solicitudes | 0 excepciones
90      public ActionResult Edit(int id)
91      {
92          try
93          {
94              // TODO: Add update logic here
95              glacier glacier= db.glaciers.Single(i => i.idGlacier == id);
96              UpdateModel(glacier);
97              db.SaveChanges();
98              return RedirectToAction("Index");
99          }
100         catch
101         {
102             return View();
103         }
104     }
```

Código 2.18 Método `Edit` implementado con `POST`

En el Código 2.19 se explica el método `Delete`, perteneciente al Controlador `glaciersController`. El método recibe como parámetro de entrada el `id` del registro de glaciar que se desea eliminar (línea 107). Se realiza una consulta a la tabla `glaciers` de la base de datos para obtener el registro de glaciar cuyo `id` sea igual al parámetro de entrada del método. El objeto retornado se almacena en una variable de tipo `glacier` (línea 109). Dado que los registros de la tabla `glacier` no deben ser eliminados de la base de datos para evitar problemas de consistencia, simplemente se cambia el estado de su atributo `stateR` al valor `False` (línea 110), con lo que se desactiva su uso en el resto del sistema. Se guardan los cambios realizados en la base de datos haciendo uso de la

instancia de la Clase Contexto (línea 111). Finalmente, se redirige a otra acción en el Controlador (línea 112).

```
106         // GET: glaciers/Delete/5
            0 referencias | 0 solicitudes | 0 excepciones
107         public ActionResult Delete(int id)
108         {
109             glacier glacier= db.glaciers.Single(i => i.idMountain == id);
110             glacier.stateR = false;
111             db.SaveChanges();
112             return RedirectToAction("Index");
113         }
```

Código 2.19 Método `Delete`

En el Código 2.20 se explica el método `ComputeHash256`, perteneciente al Controlador `systemUsersController`. Este método se encarga del cifrado de la contraseña de usuario y es utilizado, por ejemplo, al crear un nuevo registro de tipo `systemUser`, donde el usuario ingresa su contraseña, pero esta debe cifrarse antes de almacenarse en la base de datos. El método recibe como parámetro de entrada una cadena de texto (línea 265). El algoritmo de cifrado utilizado es SHA256[32]. Se llama al método `SHA256.Create()` para generar una instancia de la clase `SHA256` y se lo asigna a la variable respectiva (línea 267).

```
            4 referencias | 0 solicitudes | 0 excepciones
265         public string ComputeHash256(string password) //Allows to compute the hash function of the password and returns it as a string
266         {
267             SHA256 sha256 = SHA256.Create();
268             byte[] bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
269             StringBuilder builder = new StringBuilder();
270             for (int i = 0; i < bytes.Length; i++)
271             {
272                 builder.Append(bytes[i].ToString("x2"));
273             }
274             return builder.ToString();
275         }
```

Código 2.20 Método `ComputeHash256`

Posteriormente se calcula la función hash de la cadena de texto de entrada y se la almacena en un arreglo de `bytes` (línea 268). Para obtener la cadena de caracteres generada por la función hash, se construye un objeto de la clase `StringBuilder` para ir añadiendo cada carácter de la función hash a dicho objeto (línea 269). Luego, se itera a

[32] SHA256: Es una función hash que permite cifrar una cadena de texto. Al calcular la función hash se obtiene como salida una cadena de 256 bits, sin importar el tamaño de la cadena entrante.

través del arreglo de `bytes`, convirtiendo cada elemento de dicho arreglo a una cadena de texto y añadiéndolo uno a uno al objeto de la clase `StringBuilder` creado anteriormente (líneas 270 a 273). Finalmente, se convierte este objeto a una cadena de texto para que el método lo pueda retornar (línea 274). La contraseña cifrada se almacena en la base de datos, en caso de que se necesite recuperarla se generará una nueva contraseña.

En el Código 2.21 se expone el método `Login`, perteneciente al Controlador `systemUsersController`. El método debe tener el atributo [`HttpPost`] (línea 193) para indicar que será implementado con `POST`. Se recibe como parámetro de entrada un objeto de la clase `systemUser` (línea 194).

```
193     [HttpPost]
        0 referencias | 0 solicitudes | 0 excepciones
194     public ActionResult Login(systemUser u) //Allows users start a session
195     {
196         try
197         {
198             //if (ModelState.IsValid)
199             //{
200                 string password = ComputeHash256(u.passwordU);
201                 var user = db.systemUsers.Where(a=>a.email.Equals(u.email) && a.passwordU.Equals(password)).FirstOrDefault();
202                 //var user = db.systemUsers.Where(a => a.email.Equals(email)).FirstOrDefault();
203                 if (user != null)
204                 {
205                     if (user.idUserState == 1) //Activo
206                     {
207                         Session["idSystemUser"] = user.idSystemUser;
208                         Session["email"] = user.email;
209                         Session["name"] = user.firstName;
210                         Session["privilege"] = user.userPrivilege;
211                         Session["idUserPrivilege"] = user.idUserPrivilege;
212                         Session["activeSession"] = true;
213                         Session["error"] = null;
214                         return RedirectPermanent("Index");
215                         //return RedirectToAction("UserDashboard");
216                     }
217                     else if (user.idUserState == 3) //Pendiente
218                     {
219                         Session["error"] = "Your account is not activated yet, please wait for an approval";
220                         return RedirectToAction("Login");
221                     }
222                     else //Inactivo
223                     {
224                         Session["error"] = "Invalid Email or Password";
225                         return RedirectToAction("Login");
226                     }
227                 }
228                 else
229                 {
230                     Session.Add("error", "Invalid Email or Password");
231                     return RedirectToAction("Login");
232                 }
233             // }
234             // return View();
235         }
236         catch (Exception e)
237         {
238             Session.Add("error", " ERROR! Please contact marcos.villacis@epn.edu.ec or thomas.condom@ird.fr");
239             return View();
240         }
241     }
```

Código 2.21 Método `Login`

Luego, se realiza una consulta a la tabla `systemUsers` de la base de datos para buscar un registro en el que el valor de los atributos `email` y `passwordU` coincida con el valor de

los atributos del objeto pasado como parámetro de entrada, y se almacena dicho registro en el objeto `user`.

Es importante señalar que, dado que las contraseñas almacenadas en la base de datos están cifradas, es necesario cifrar la cadena de texto correspondiente al atributo `passwordU` del objeto pasado como parámetro de entrada antes de compararla con los valores almacenados en la respectiva tabla (líneas 200 y 201).

A continuación, se realiza una validación para verificar si el objeto `user` es nulo, lo cual indicaría que la búsqueda en la tabla `systemUsers` no encontró coincidencias, es decir, que no existe un usuario registrado con dichos valores en sus atributos `email` y `passwordU`. En caso de ser nulo, se genera un mensaje indicando que no existe ese usuario registrado en el sistema (líneas 228 a 232). Caso contrario, se verifica el estado de la cuenta de usuario, si su valor es `Active`, entonces se procede con el inicio de sesión. De no ser así, se genera un mensaje indicando que el estado actual de la cuenta de usuario no le permitirá iniciar sesión (líneas 205 a 226).

El Código 2.22 presenta el método `SendEmailForConfirmation`, perteneciente al Controlador `systemUsersController`. Este método se utiliza cuando un usuario crea una nueva cuenta. Debido a que es necesario que el Administrador del sistema confirme la creación de la cuenta y la active, se envía un correo electrónico con los datos del usuario que se registró.

```
1 referencia | 0 solicitudes | 0 excepciones
277     public string sendEmailForConfirmation(systemUser newUser)
278     {
279         string cuerpo = "A new user has registered. A status of pending has been " +
280             "assigned until you confirm the registration. " +
281             "User`s information summary: \n Name: " + newUser.firstName
282             + " \n Last Name: " + newUser.lastName + "\n Email: "
283             + newUser.email + " \n\n To review complete information please login in the system";
284
285         correo.Subject = "New User Confirmation Required";
286         correo.SubjectEncoding = System.Text.Encoding.UTF8;
287         correo.To.Add("javier.armas@epn.edu.ec");
288         correo.Body = cuerpo;
289         correo.BodyEncoding = System.Text.Encoding.UTF8;
290         try
291         {
292             protocolo.Send(correo);
293             return "OK";
294         }
295         catch(SmtpException e)
296         {
297             return e.ToString();
298         }
299     }
```

Código 2.22 Método `SendEmailForConfirmation`

El método recibe como parámetro de entrada un objeto de la clase `systemUser` que contiene toda la información del usuario registrado (línea 277). Se crea una variable de tipo `string` con un resumen del texto que será enviado al correo electrónico del Administrador del sistema (líneas 279 a 283). Se define el asunto del correo que se va a enviar (línea 285), para esto, anteriormente se debió declarar un objeto de la clase `MailMessage`. Se realiza la codificación del asunto del correo mediante UTF8[33] (línea 286). Después se especifica el destinatario del correo (línea 287). Luego se establece el cuerpo del correo (línea 288), que contendrá el texto almacenado en la variable creada anteriormente. Se realiza la codificación del texto que irá en el cuerpo del correo (línea 289). Finalmente, se envía el correo con los parámetros indicados (línea 292).

El método `PasswordRecovery` perteneciente al Controlador `systemUsersController` se expone en el Código 2.23. Este método permite al usuario que ha olvidado su contraseña recibir una nueva contraseña temporal en su correo electrónico que le puede servir para iniciar sesión y posteriormente cambiarla por otra contraseña.

```
0 referencias | 0 solicitudes | 0 excepciones
public ActionResult passwordRecovery(string email)
{
    if (ModelState.IsValid)
    {
        var user = db.systemUsers.Where(a => a.email.Equals(email)).FirstOrDefault();
        if (user != null)
        {
            if(user.idUserState == 1)
            {
                //Generate temporary password
                int max = 999999999;
                int min = 9999999;
                Random r = new Random();
                int num = r.Next(min, max);
                string temporaryPassword = ComputeHash256(Convert.ToString(num));
                user.passwordU = temporaryPassword;
                sendEmailForRecovery(user, temporaryPassword, num);
                db.SaveChanges();
                Session.Add("error", "A temporary password was sent to your email. Use this password to log in and do not forget to change it!");
            }
            else
            {
                Session.Add("error", "Your user has no been activated yet!");
            }

            return RedirectToAction("Login");
        }
        else
        {
            Session.Add("error", "User does not exist! Please insert a valid email or create an account");
            return RedirectToAction("passwordRecovery");
        }
    }
    return View();
}
```

Código 2.23 Método `PasswordRecovery`

[33] UTF8: Es un formato de codificación de caracteres Unicode utilizando símbolos de longitud variable.

El método recibe como parámetro de entrada el correo electrónico del usuario que olvidó su contraseña (línea 307). Considerando que únicamente las cuentas de usuario que se encuentran en estado `Active` pueden iniciar sesión, solo dichas cuentas pueden solicitar una contraseña temporal en caso de olvido, para esto se realizan algunas validaciones. Para generar una contraseña temporal, se genera un número aleatorio comprendido entre un valor mínimo y un valor máximo (líneas 317 a 320). Este número corresponderá a la nueva contraseña, por lo que debe ser cifrado antes de almacenarlo en la base de datos (línea 321). Una vez cifrado, se modifica el registro respectivo en la base de datos, se envía el correo electrónico con la nueva contraseña temporal al usuario y se guardan los cambios en la base de datos (líneas 322 a 324). También se muestra un mensaje indicando el envío del correo con la nueva contraseña (línea 325).

El Código 2.24 muestra el método `GetStationList` perteneciente al Controlador `enteredDataController`. Este método permite obtener una lista de registros pertenecientes a la tabla `stations`, para luego desplegarlos en un `DropDownList` en función de lo que se ha seleccionado en otro `DropDownList` previamente. El método recibe como parámetro de entrada un `id` correspondiente al valor seleccionado en el primer `DropDownList`. El tipo de acción del método es `JsonResult`, por lo que retornará información en formato JSON (línea 102). Luego se realiza una consulta a la tabla `stations` de la base de datos para buscar los registros cuyo atributo `idMeasurementType` contenga un valor igual al valor del parámetro de entrada. La información retornada es almacenada en una lista de tipo `station` (línea 105). Finalmente, el método retorna el texto en formato JSON con los registros de las estaciones que pertenecen a un determinado tipo de medida (línea 106).

```
                0 referencias | 0 solicitudes | 0 excepciones
102             public JsonResult GetStationList(int idMType) //Obtiene la lista de estaciones para el Dropdownlist
103             {
104                 db.Configuration.ProxyCreationEnabled = false;
105                 List<station> stations = db.stations.Where(x => x.idMeasurementType == idMType).ToList();
106                 return Json(stations, JsonRequestBehavior.AllowGet);
107             }
108
```

Código 2.24 Método `GetStationList`

En el Código 2.25 se describe parte del método `Search` perteneciente al controlador `enteredDataController`. Este método permite realizar una búsqueda de información con base en los criterios seleccionados por el usuario en la Vista correspondiente. La

porción de código mostrado permite buscar información de meteorología. Para realizar una búsqueda de información, el usuario debe especificar ciertos parámetros, como país (`projectCountry`), montaña (`mountain`), glaciar (`glacier`), tipo de dato (`dataType`), tipo de medida (`measurementType`), estación (`station`), variable (`variable`) y año (`year`). Todos estos datos son enviados como parámetro de entrada al método `Search` a través de un objeto de la clase `EnteredDataExt` (línea 472).

```
0 referencias | 0 solicitudes | 0 excepciones
472  public ActionResult Search(EnteredDataExt u)
473  {
474      int year = u.year;//Convert.ToDateTime("2007-01-01");
475      string variable = u.variable;
476
477      List<meteorologicalData> searchedData = (from doc in db.enteredDatas
478                                               join data in db.meteorologicalDatas on doc.idEnteredData equals data.idEnteredData
479                                               where doc.idStation == u.idStation && (SqlFunctions.DatePart("year", doc.startDate) == year || SqlFunctions.DatePart("year", doc.endDate) == year)
480                                               && SqlFunctions.DatePart("year", data.dateData) == year  //&& Convert.ToDouble(data.Get<meteorologicalData>(variable).ToString()) != -6999
481                                               select data).ToList();
482
483      List<FilteredData> summary = new List<FilteredData>();
484      foreach (var item in searchedData)
485      {
486          if (Double.Parse(item.Get<double>(variable).ToString()) != -6999)
487          {
488              FilteredData aux = new FilteredData()
489              {
490                  key1 = item.dateData,
491                  Key2 = Double.Parse(item.Get<double>(variable).ToString())
492              };
493              summary.Add(aux);
494          }
495      }
496      IEnumerable<FilteredData> result;
497      if (variable == "precipitationAmount") //Para precipitaciones se suma (valor acumulado)
498      {
499          result = from f in summary
500                   group f by f.key1.Date into g
501                   select new FilteredData { key1 = g.Key, Key2 = g.Sum(x => Convert.ToDouble(x.Key2.ToString())) };
502
503
504      }
505      else //Para el resto de variables se promedia
506      {
507          result = from f in summary
508                   group f by f.key1.Date into g
509                   select new FilteredData { key1 = g.Key, Key2 = g.Average(x => Convert.ToDouble(x.Key2.ToString())) };
510
511      }
512      Session.Add("resultInfo", result);
513      Session.Add("variableInfo", variable);
514      generateGraphics(result, variable);
515
516      //ViewBag.imagen = image;
517      return View("ShowResults", result);
518  }
```

Código 2.25 Método `Search`

Posteriormente, mediante LINQ se consulta en la base de datos los registros de meteorología que cumplen con los parámetros solicitados. Los registros retornados son asignados a una lista de tipo `meteorologicalData` (líneas 477 a 481). Por petición de los miembros del proyecto LMI GREAT ICE, los resultados de la búsqueda deben mostrarse en un diagrama temporal. El diagrama graficará el valor de la variable especificada en distintos momentos de tiempo.

Los registros de meteorología indican que cada variable es medida cada 30 minutos. Cada documento de meteorología contiene los valores medidos para distintas variables a lo largo

de un año. Debido a esto, no es necesario contar con toda la información que devuelve la consulta LINQ, sino que se debe cortar las tuplas para que cada una contenga únicamente dos columnas, fecha y valor, que son las que se graficarán.

Por esta razón, se hace uso de la clase `FilteredData` para generar un resumen que contenga únicamente la información a graficar. Esto se hace iterando cada uno de los elementos de la lista de tipo `meteorologicalData` y guardando solo los valores de fecha y valor correspondientes a cada elemento, en una lista de tipo `FilteredData` (líneas 483 a 495). A pesar de que cada variable es medida cada 30 minutos, se solicitó que se genere un resultado diario para cada variable, y que sea este resultado el que se grafique para todo el año. Para esto existen dos opciones: En las variables de tipo `precipitationAmount`, el resultado diario se obtiene sumando los valores recogidos cada 30 minutos (líneas 497 a 504). En todas las demás variables, el resultado diario se obtiene promediando los valores recogidos cada 30 minutos (líneas 505 a 511). Finalmente, se genera el gráfico temporal con los resultados obtenidos (línea 514).

El Código 2.26 explica el método `GenerateGraphics` perteneciente al controlador `enteredDataController`. Este método es el encargado de generar las gráficas temporales con los resultados de la búsqueda realizada por el usuario.

```
777     public void generateGraphics(IEnumerable<FilteredData> data, string variable, EnteredDataExt searchedInfo)
778     {
779         List<DateTime> dates = new List<DateTime>();
780         List<double?> values = new List<double?>();
781         try
782         {
783             foreach (var item in data)
784             {
785                 dates.Add(item.key1);
786                 values.Add(item.Key2);
787             }
788         }
789         catch (Exception ex) { }
790         new Chart(width: 600, height: 400, theme: ChartTheme.Green).AddTitle("Results").
791             AddSeries(chartType: "Line", xValue: dates, yValues: values).AddLegend(variable, null).
792             Save("~/Content/images/hola.jpeg", "jpeg");
793     }
```

Código 2.26 Método `GenerateGraphics`

Se recibe como parámetro de entrada una colección de tipo `FilteredData` que contiene los resultados de la búsqueda, y la variable a graficar (línea 777). Para generar el gráfico, deben primero construirse las series temporales que irán ubicadas en el eje X y en eje Y. Para ello, se itera a través de la colección y se asignan los valores de las fechas a la

variable `dates`, mientras que los valores de la medida se asignan a la variable `values` (líneas 783 a 787). Finalmente, se grafica las series y se configuran ciertos parámetros que tendrá el gráfico (líneas 790 a 792).

El método `ExportCSV` perteneciente al Controlador `enteredDataController` se expone en el Código 2.27. Este método es el encargado de construir un documento en formato CSV con los resultados de la búsqueda de información y descargarlo. Para ello se hizo uso de la librería CsvExport, que fue instalada mediante el gestor de paquetes NuGet. Una vez instalada la librería se creó un método de acción de tipo `FileResult` para retornar el contenido de un archivo (línea 542). Dado que esta acción se va a ejecutar al presionar un botón de descarga en la Vista, el método debe llevar el atributo [`HttpPost`] (línea 541). Inicialmente se establece el nombre que tendrá el archivo a descargar, en este caso se asignará el nombre de la variable buscada (línea 544). Se generará una nueva instancia de `CsvExport` (fila 546). Luego se debe iterar a través de la colección de resultados obtenidos para ir añadiendo en cada iteración, una fila al documento con los valores correspondientes a las variables `Date` y `Value` en las columnas respectivas (filas 548 a 553). Finalmente, el método retorna el archivo en formato CSV (línea 554).

```
541         [HttpPost]
            0 referencias | 0 solicitudes | 0 excepciones
542         public FileResult exportCSV()
543         {
544             var filename = Convert.ToString(Session["variableInfo"]);
545             var results = (IEnumerable<FilteredData>)(Session["resultInfo"]);
546             var myCSV = new Jitbit.Utils.CsvExport();
547
548             foreach (var iter in results)
549             {
550                 myCSV.AddRow();
551                 myCSV["Date"] = iter.key1.Date;
552                 myCSV["Value"] = iter.Key2;
553             }
554             return File(myCSV.ExportToBytes(), "application/csv", filename);
555         }
556     }
```

Código 2.27 Método `ExportCSV`

El Código 2.28 detalla el método `UploadFile` perteneciente al Controlador `AsynController`. El Controlador `AsynController` contiene todas las tareas que se encargan de cargar nueva información de los documentos a la base de datos, independientemente de si los documentos son de meteorología, glaciología o hidrología. Tomando en cuenta que los documentos contienen una gran cantidad de filas, y que el almacenamiento de su información en el sistema tarda un tiempo considerable, se decidió que la carga de información debe hacerse de forma asincrónica. Es decir, que al momento

de cargar la información debe liberarse el hilo de ejecución del proceso para que se puedan realizar otras tareas al mismo tiempo.

```
28      [HttpPost]
        0 referencias | 0 solicitudes | 0 excepciones
29      public ActionResult UploadFile(HttpPostedFileBase fileToCheck, enteredData enteredData) //HttpPostedFileBase FileUpload
30      {
31          if (Request != null)//control generico de existencia de archivo
32          {
33              try
34              {
35                  if (fileToCheck.ContentLength > 0) //TAMAÑO DEL ARCHIVO MAYOR A 0 BYTES
36                  {
37                      if (validateFileExtension(fileToCheck)) //extension csv
38                      {
39                          StreamReader csvReader = new StreamReader(fileToCheck.InputStream);
40                          Holder hold = validateFileStructure(csvReader, enteredData); //validación de estructura de archivo
41                          if (hold.status == true)
42                          {
43                              int idUser = int.Parse(Session["idSystemUser"].ToString());
44                              var fileName = Path.GetFileName(fileToCheck.FileName);
45                              var path = Path.Combine(Server.MapPath("~/UploadedFiles/TemporaryFiles"), fileName);
46                              fileToCheck.SaveAs(path);
47                              string finalName = "ORIGINAL-" + fileName;//Path.GetFileName(fileToCheck.FileName);
48                              string finalPath = Path.Combine(Server.MapPath("~/UploadedFiles"), finalName);
49                              BackgroundJob.Enqueue(() => saveFileData(fileName,path, finalPath,enteredData, idUser));
50                          }
51                          else
52                          {
53                              Session["error"] = hold.message;
54                              return RedirectToAction("UploadFile");
55                          }
56                      }
57                      else
58                      {
59                          Session["error"] = "The file must have an extension .csv";
60                      }
61                      return RedirectToAction("UploadFile");
62                  }
63              }
64              catch(Exception ex)
65              {
66                  Session["error"] = ex.Message;//"File upload failed!" + " " + "Please verify that there are no NULL fields";
67                  return RedirectToAction("UploadFile");
68              }
69          }
70          return RedirectToAction("UploadFile");
71      }
```

Código 2.28 Método `UploadFile`

En un inicio, se implementaron métodos asíncronos para realizar la carga de información. Sin embargo, esto no funcionó, pues el hilo de ejecución esperaba la culminación de la instrucción para retornar una Vista, lo cual no permitía ejecutar otras acciones simultáneamente.

Para cumplir con el objetivo se utilizó la librería HangFire [47], mediante la cual se agregaron hilos en segundo plano para la ejecución de los procesos.

`UploadFile` es el método principal en el Controlador `AsynController`, pues es el encargado de cargar la información de los archivos al sistema, así como de guardar el

archivo original en una carpeta, de forma asincrónica. Este método contiene el atributo [`HttpPost`] para indicar que es una acción que será implementada utilizando `POST` (línea 28).

Recibe como parámetros de entrada una instancia de la clase `HttpPostedFileBase` que contiene el archivo que se desea cargar, y una instancia de la clase `enteredData`, que contiene información acerca del documento a cargar (línea 29). Antes de cargar el archivo se realizan una serie validaciones que permiten que no haya errores al ejecutar la acción. Por ejemplo, se valida si el archivo existe (línea 31); se verifica que el archivo no esté vacío y su tamaño sea mayor a 0 bytes (línea 35); se valida que la extensión del archivo sea .csv (línea 37). Una vez cumplidas estas validaciones, se declara un objeto de la clase `StreamReader` para leer los caracteres del archivo (línea 39). Al leer los caracteres del documento, se realiza una validación de la estructura del archivo (línea 40). Si es que la validación cumple con un formato establecido, entonces se guarda el archivo original en una carpeta, y la información contenida en el archivo se almacena en la base de datos (líneas 41 a 50). En caso de no cumplirse las validaciones, la información no se guarda y se muestran mensajes indicando el error. En la línea 49 se llama al método `saveFileData` para efectuar el almacenamiento de la información y del documento, pero antes se crean hilos en segundo plano que permitirán que estas acciones se ejecuten sobre ellos, liberando el hilo principal para que se puedan ejecutar otras tareas mientras se carga la información al sistema.

2.8.3.3 Vistas

Al momento de crearse un nuevo Controlador en la carpeta `Controllers`, se crea también una carpeta con el nombre del Controlador dentro de la carpeta `Views`. Cada carpeta puede contener una o más Vistas, en función de las necesidades.

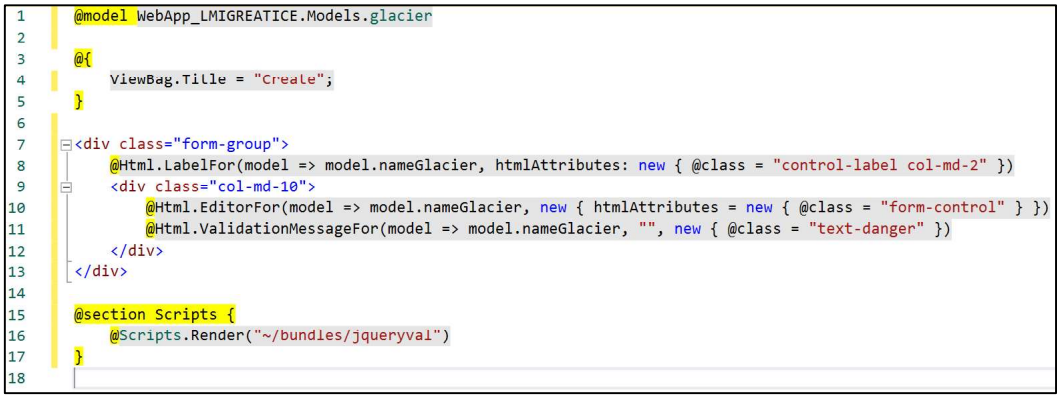
Las Vistas hacen uso de código HTML, CSS y JavaScript. El código HTML provee la estructura de la Vista. El código CSS se encarga de aplicar estilos y plantillas para que la Vista tenga una buena apariencia. El código JavaScript añade interactividad a la Vista.

También se utiliza la sintaxis RAZOR para juntar código C# con HTML, en la cual se usa el símbolo `@` para realizar la transición entre códigos.

La organización general del código en una Vista se detalla en la Figura 2.28. Primeramente, se especifica la entidad del Modelo que se va a utilizar en la Vista (línea 1). La sentencia `@model` permite acceder al Modelo y sus propiedades.

En segundo lugar, se encuentra un bloque de código que utiliza la sintaxis RAZOR para indicar el título de la página (líneas 3 a 5). Luego, se tiene un bloque de código HTML que define la estructura que tendrá la Vista. Este bloque puede hacer uso de la sintaxis RAZOR para acceder a información del Modelo y presentarla en la Vista. En este caso, se utiliza la sintaxis RAZOR al definir *HTML Helpers* para desplegar información del Modelo en la Vista (líneas 8 a 11).

Finalmente se tiene un bloque de código JavaScript que permite agregar interactividad mediante la renderización de *scripts* (líneas 15 a 17). Pueden utilizarse librerías como jQuery para facilitar el desarrollo en esta parte del código.



```
1  @model WebApp_LMIGREATICE.Models.glacier
2
3  @{
4      ViewBag.Title = "Create";
5  }
6
7  <div class="form-group">
8      @Html.LabelFor(model => model.nameGlacier, htmlAttributes: new { @class = "control-label col-md-2" })
9      <div class="col-md-10">
10         @Html.EditorFor(model => model.nameGlacier, new { htmlAttributes = new { @class = "form-control" } })
11         @Html.ValidationMessageFor(model => model.nameGlacier, "", new { @class = "text-danger" })
12     </div>
13 </div>
14
15 @section Scripts {
16     @Scripts.Render("~/bundles/jqueryval")
17 }
18
```

Figura 2.28 Organización del código en una Vista

En el sistema desarrollado, la Vista `Search.cshtml` que se encuentra en la carpeta `enteredData` tiene una importancia particular. En esta Vista se realiza la implementación del mapa interactivo y sus marcadores.

Para implementar un mapa interactivo utilizando la librería Leaflet, debe iniciarse añadiendo un elemento de tipo `div` a la Vista y asignándole un `id`.

Claramente, esto se debe realizar en un bloque de código HTML, como se observa en el Código 2.29. Aquí, se indica también la altura y ancho del mapa.

```
88          <div id="map" style="height:350px;width:650px"></div>
```

Código 2.29 Añadiendo un elemento `div` para el mapa interactivo

Dado que Leaflet es una librería de JavaScript, el resto de la implementación debe realizarse en el bloque de código JavaScript, como se observa en el Código 2.30. Es necesario crear una instancia del mapa y configurar parámetros como las coordenadas para centrarlo y el nivel de acercamiento (*zoom*) (línea 156). Luego se añade la capa proveniente de OpenStreetMap al mapa (líneas 157 a 167), donde se puede especificar el nivel de acercamiento máximo posible.

```
156 var mymap = L.map('map').setView([-22.30, -58], 2.2);
157 L.tileLayer
158     ('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access
159     token = pk.eyJ1IjoiamFhbjkwIiwiYSI6ImNqczYyeTBwZzBod2E0YW1zNWxlZDUxeWkifQ.73osagGl_LLhZT7M1aGvTA',
160     {
161         attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors,
162             < a href="https://creativecommons.org/licenses/by-sa/2.0/" > CC - BY - SA</a>, Imagery ©
163         < a href = "https://www.mapbox.com/" > Mapbox</a > ',
164     maxZoom: 18,
165     id: 'mapbox.streets',
166     accessToken: 'pk.eyJ1IjoiamFhbjkwIiwiYSI6ImNqczYyeTBwZzBod2E0YW1zNWxlZDUxeWkifQ.73osagGl_LLhZT7M1aGvTA'
167 }).addTo(mymap);
```

Código 2.30 Código para implementar el mapa interactivo

Un ejemplo de Vista cuyo código es relativamente similar al que contienen el resto de las Vistas, se presenta en el Código 2.31.

```
1  @model WebApp_LMIGREATICE.Models.glacier
2
3  @{
4      ViewBag.Title = "Create";
5  }
6
7  <h2 class="text-center">Create new glacier</h2>
8
9  @using (Html.BeginForm())
10 {
11     @Html.AntiForgeryToken()
12
13     <div class="form-horizontal">
14         <hr />
15         @Html.ValidationSummary(true, "", new { @class = "text-danger" })
16         <div class="form-group">
17             @Html.LabelFor(model => model.nameGlacier, htmlAttributes: new { @class = "control-label col-md-2" })
18             <div class="col-md-10">
19                 @Html.EditorFor(model => model.nameGlacier, new { htmlAttributes = new { @class = "form-control" } })
20                 @Html.ValidationMessageFor(model => model.nameGlacier, "", new { @class = "text-danger" })
21             </div>
22         </div>
23
24         <div class="form-group">
25             @Html.LabelFor(model => model.idMountain, "Mountain", htmlAttributes: new { @class = "control-label col-md-2" })
26             <div class="col-md-10">
27                 @Html.DropDownListFor(model => model.idMountain, ViewBag.mountains as SelectList, htmlAttributes: new { @class = "form-control" })
28                 @Html.ValidationMessageFor(model => model.idMountain, "", new { @class = "text-danger" })
29             </div>
30         </div>
```

Código 2.31 Ejemplo de Vista

La Vista en cuestión es `Create.cshtml` de la carpeta `glaciers`. Inicialmente, se indica que se trabajará con la entidad `glacier` (línea 1). Luego se hacer uso de la sintaxis RAZOR para añadir un título a la página (líneas 3 a 5). Finalmente se hace uso de `HtmlHelpers` y sintaxis RAZOR para mostrar elementos como etiquetas (`LabelFor`), cuadros de texto (`EditorFor`) y listas desplegables (`DropDownListFor`). En las listas desplegables se muestra la información almacenada en contenedores `ViewBag` (líneas 9 a 30).

2.8.3.4 Interfaces de usuario

Las Vistas contienen el código que será interpretado por el navegador en el lado del cliente para generar las interfaces de usuario respectivas. A continuación, se muestran las principales interfaces de usuario generadas. La Figura 2.29 corresponde a la interfaz de usuario de la página principal del sistema. En ella, el usuario puede realizar búsquedas de información con base en distintos criterios.

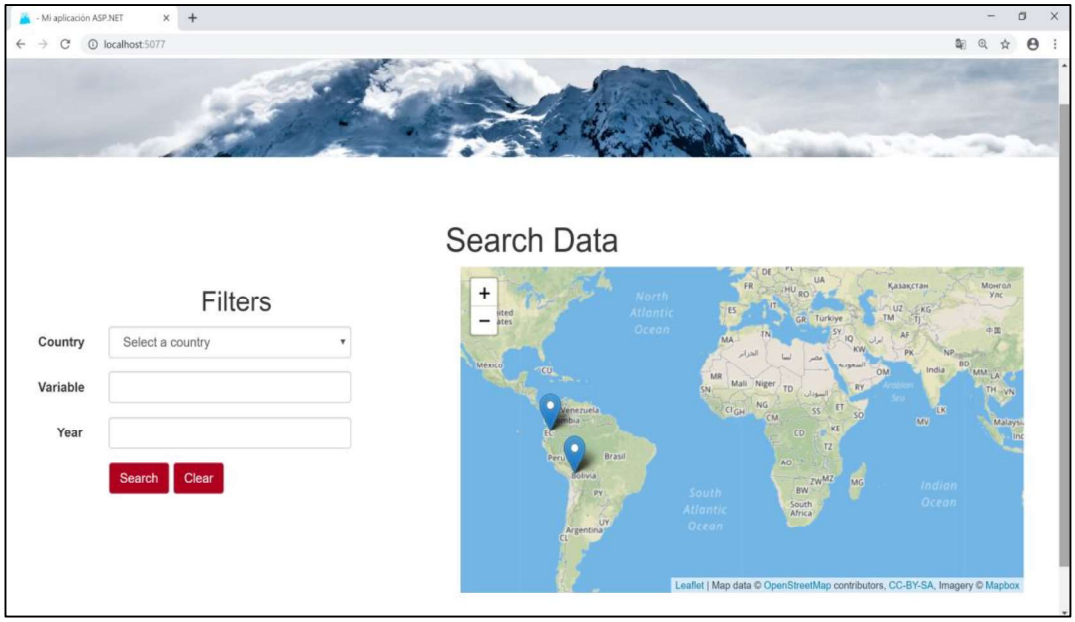


Figura 2.29 Interfaz de usuario de la página principal

En el lado izquierdo de la pantalla se muestra la sección de filtros, donde se debe seleccionar una serie de opciones para buscar información. En el lado derecho, se muestra

el mapa interactivo con los marcadores correspondientes a las estaciones que actualmente recopilan información para el proyecto LMI GREAT ICE. Al seleccionar un marcador, se despliega información sobre la estación asociada a ese marcador.

En esta pantalla también están presentes las opciones para crear una nueva cuenta de usuario y para iniciar sesión. Además, tanto en esta pantalla como en el resto, se colocó un banner en la parte superior con una imagen del volcán Antisana.

En la Figura 2.30 se muestra la interfaz de usuario de la página de resultados. En ella se presentan los resultados de la búsqueda realizada. En la parte izquierda de la pantalla, se muestra información sobre la estación que se usó en la recolección de los datos buscados. En la parte derecha de la pantalla, se expone el diagrama temporal que grafica los resultados de la búsqueda. También, esta pantalla cuenta con un botón para descargar la información de resultados en un documento con formato CSV.

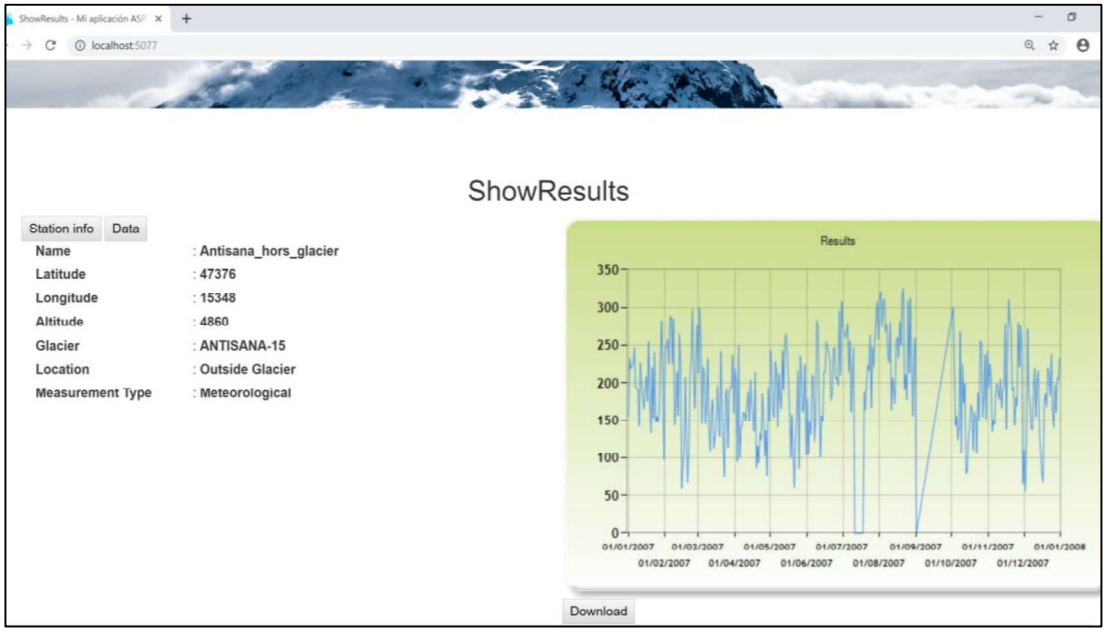


Figura 2.30 Interfaz de usuario correspondiente a la página de resultados

La Figura 2.31 corresponde a la interfaz de usuario de la página para crear una nueva cuenta de usuario. Aquí se muestran los elementos necesarios para que un usuario ingrese su información. Todos los campos son obligatorios. De esta manera, para crear una nueva cuenta, el usuario requiere ingresar su nombre, apellido, correo, contraseña, género, título, fecha de nacimiento, país e institución a la que pertenece.

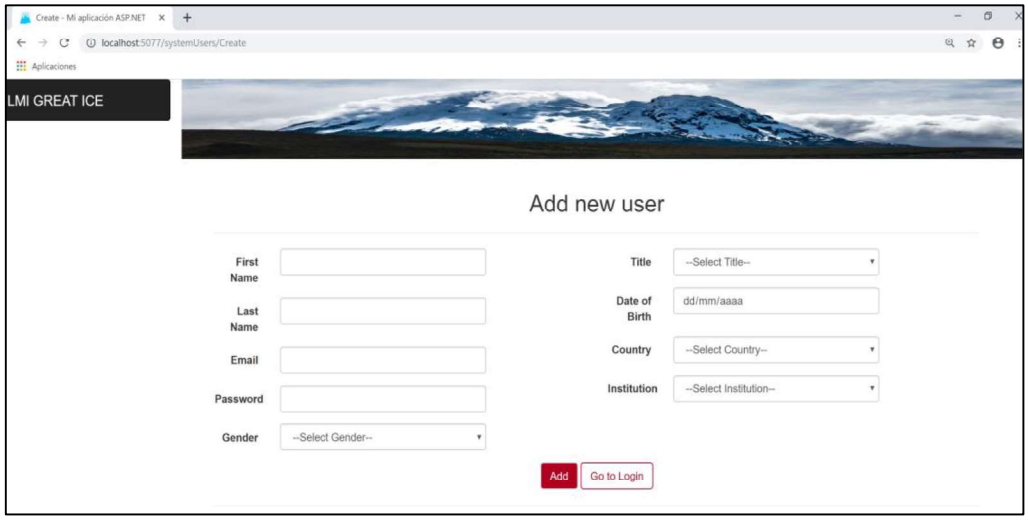


Figura 2.31 Interfaz de usuario de la página para crear una nueva cuenta

En la Figura 2.32 se muestra la interfaz de usuario correspondiente a la página de inicio de sesión. En la pantalla se presentan los dos campos que un usuario debe llenar para iniciar sesión en el sistema de forma satisfactoria. Estos dos campos son: correo electrónico y contraseña. Si el usuario posee una cuenta activa en el sistema se puede iniciar sesión, caso contrario se despliegan mensajes mencionando el error.

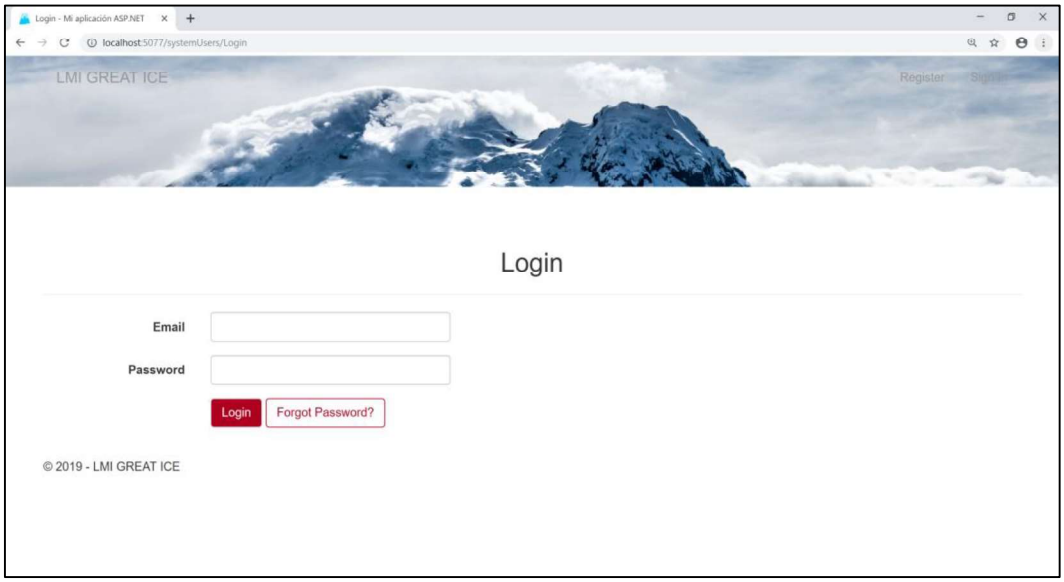


Figura 2.32 Interfaz de usuario correspondiente a la página de inicio de sesión

También, se cuenta con una opción en caso de que el usuario olvide su contraseña. Al seleccionar esta opción, se redirige a otra Vista donde el usuario debe ingresar su correo electrónico para recibir en él, una contraseña temporal.

La Figura 2.33 corresponde a la interfaz de usuario de la página a la que se llega al iniciar sesión como Administrador. En esta página, se muestra por defecto una tabla con información sobre los usuarios que poseen una cuenta en el sistema. Es decir, los usuarios almacenados en la base de datos.

También se muestran opciones para añadir nuevas cuentas de usuario, editar cuentas existentes o eliminar cuentas. En la parte izquierda de la pantalla se muestra un panel con distintos elementos. A través de este panel se consigue gestionar el sistema.

Para cada elemento seleccionado en el panel se pueden efectuar operaciones similares a las que se realizan con el elemento `Users`. En otras palabras, se permite mostrar, añadir, editar o eliminar registros de los elementos `Users`, `Mountains`, `Glacier`, `Stations`, `Institutions` y `Data`.

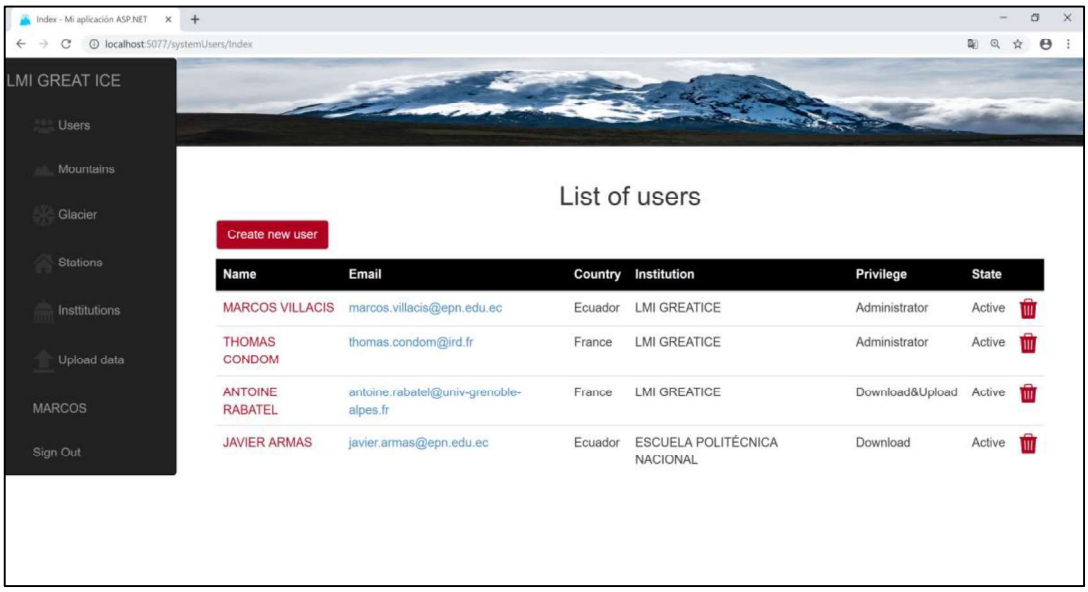


Figura 2.33 Interfaz de usuario del panel principal del Administrador

En la Figura 2.34 se muestra la interfaz de usuario de la pantalla que permite realizar la edición/actualización de información sobre una cuenta de usuario específica. Aquí, un

usuario o el Administrador del sistema pueden cambiar los datos de una cuenta y guardarlos para que los registros en la base de datos se actualicen.

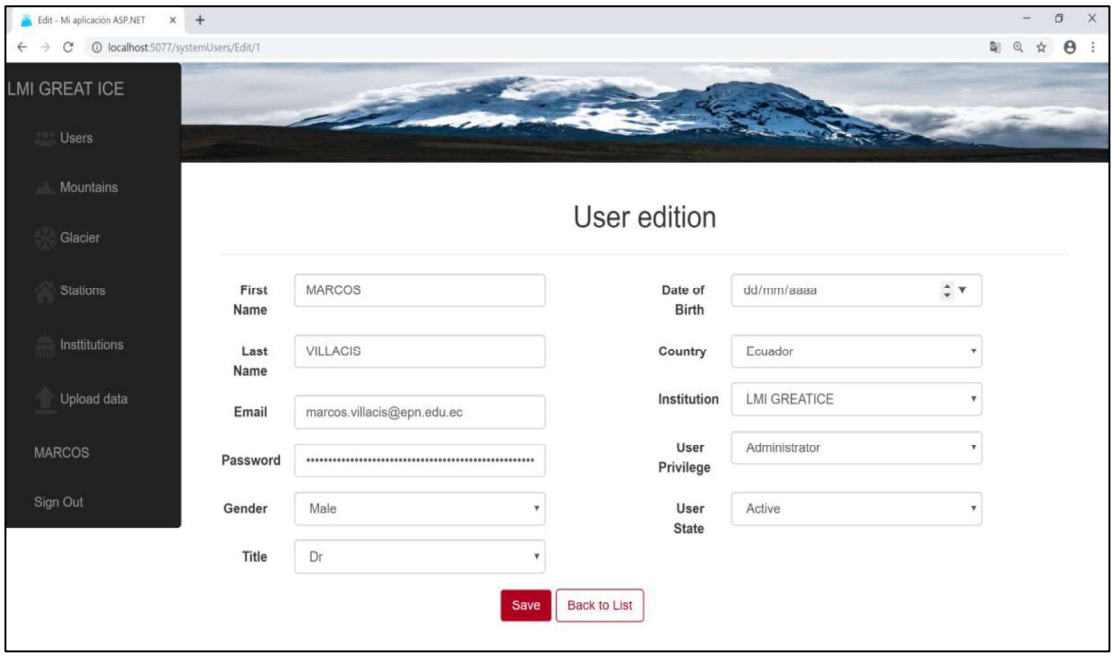


Figura 2.34 Interfaz de usuario para editar información de usuarios

La Figura 2.35 presenta la interfaz de usuario de la página que permite subir un nuevo documento al sistema.

En esta página, un usuario con el perfil adecuado (`Administrator` o `Download&Upload`) debe especificar cierta información sobre el documento que se va a cargar, como: fecha de inicio de la toma de medidas; fecha de finalización de la toma de medidas; tipo de dato; tipo de medida; y estación que recolectó la información contenida en el documento.

También se debe seleccionar desde la computadora del usuario, el documento a cargar en el sistema.

Si las distintas validaciones provistas en el código son correctas, la información que contiene el documento será almacenada en la base de datos y el archivo original del documento será guardado en una carpeta.

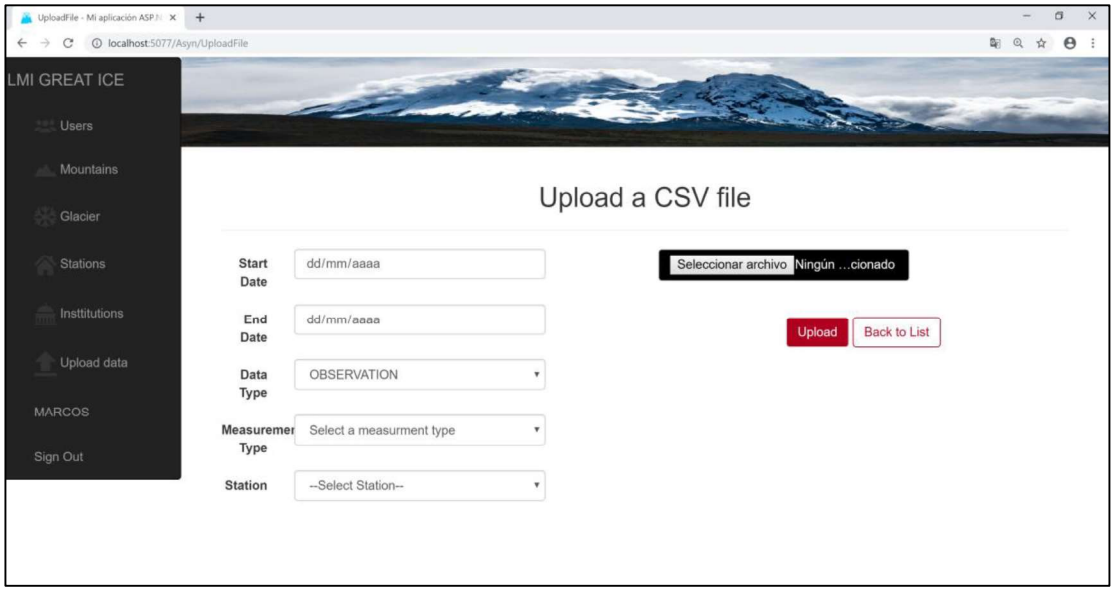


Figura 2.35 Interfaz de usuario para subir un nuevo documento

2.8.4 DESPLIEGUE

Para el despliegue del sistema se hizo uso de la plataforma Microsoft Azure, que provee servicios de computación en la nube como el alojamiento de sitios web utilizando IIS. Para esto es necesario tener una cuenta en la plataforma.

En Azure se alojará tanto la base de datos como la aplicación web.

Inicialmente, se creó y configuró el recurso Azure App Service, mediante el cual se genera el perfil de publicación para la aplicación web, tal como se puede observar en la Figura 2.36 . Aquí se especifica, entre otras cosas, el nombre de instancia que se usará en la URL para publicar la aplicación en Internet.

Una vez creado el recurso, debe seleccionarse la opción `Obtener perfil de publicación`, mediante la cual se descargará un archivo con el perfil creado, como se muestra en la Figura 2.37.

Luego, en el explorador de soluciones de Visual Studio, clic derecho en la solución y seleccionar la opción `Publicar`. En la ventana que se despliega, marcar la opción `Seleccionar existente`, y luego clic en la opción `Importar perfil`, como puede observarse en la Figura 2.38. Se abrirá una nueva ventana donde se deberá seleccionar el archivo descargado previamente. Por último, se selecciona la opción `Publicar`.

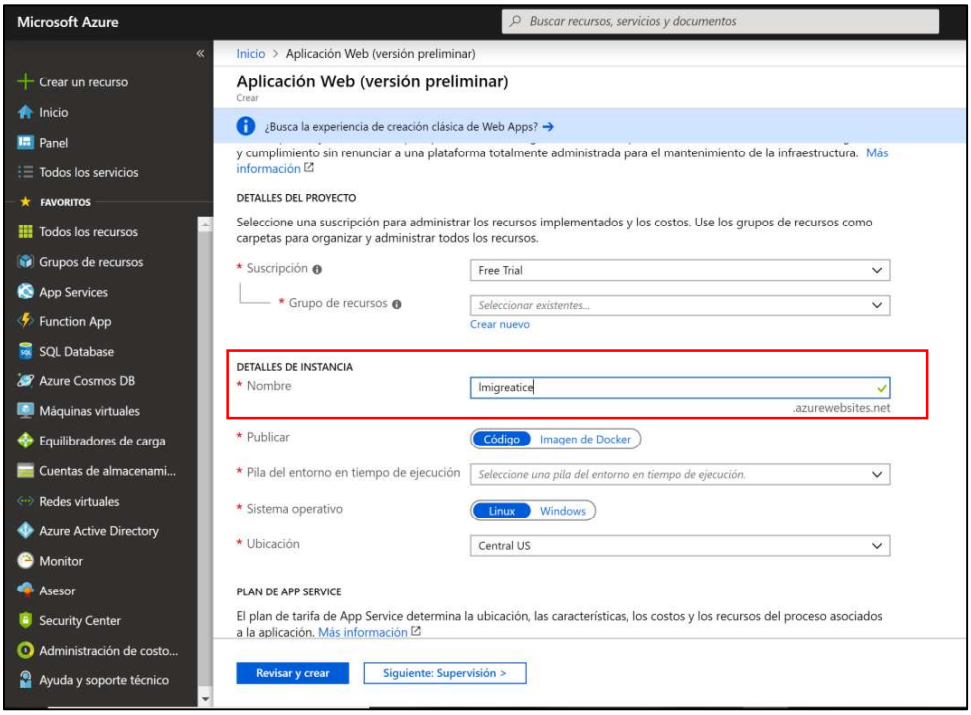


Figura 2.36 Creación del perfil de publicación de la aplicación web en Azure

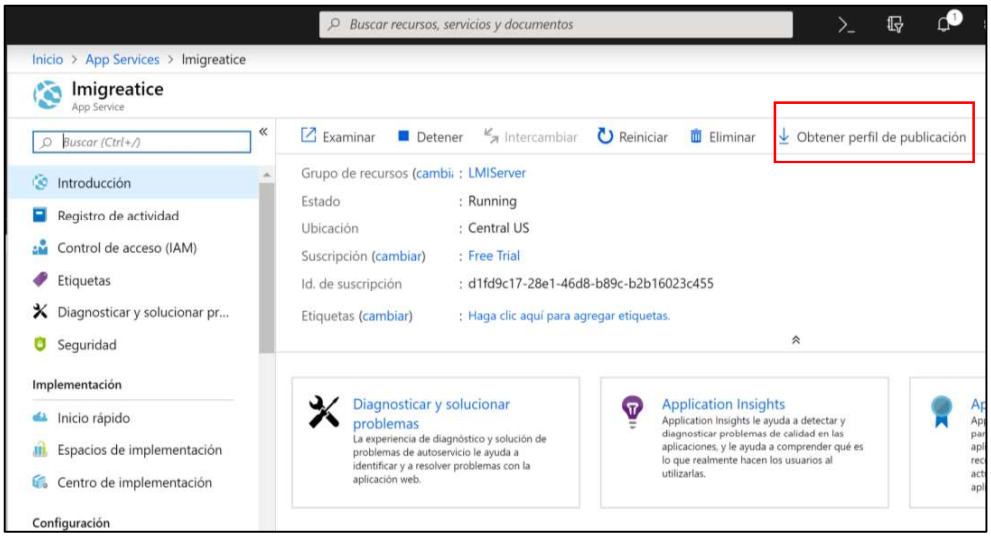


Figura 2.37 Descarga del perfil de publicación creado

Posteriormente se debe crear la base de datos en Microsoft Azure, tal como se observa en la Figura 2.39, donde se debe indicar el nombre de la base de datos, el nombre del servidor, entre otras cosas. En caso de no tener un servidor, es necesario crearse uno, para lo cual debe seleccionarse la opción `Crear nuevo`. En el panel desplegado, debe indicarse el

nombre del servidor y las credenciales de acceso al mismo, tal como se presenta en la Figura 2.40.

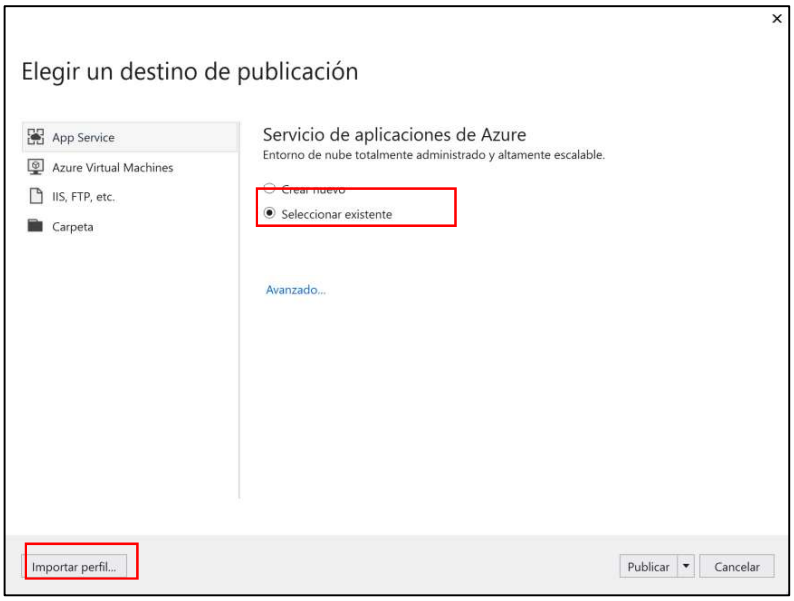


Figura 2.38 Publicación de la aplicación web

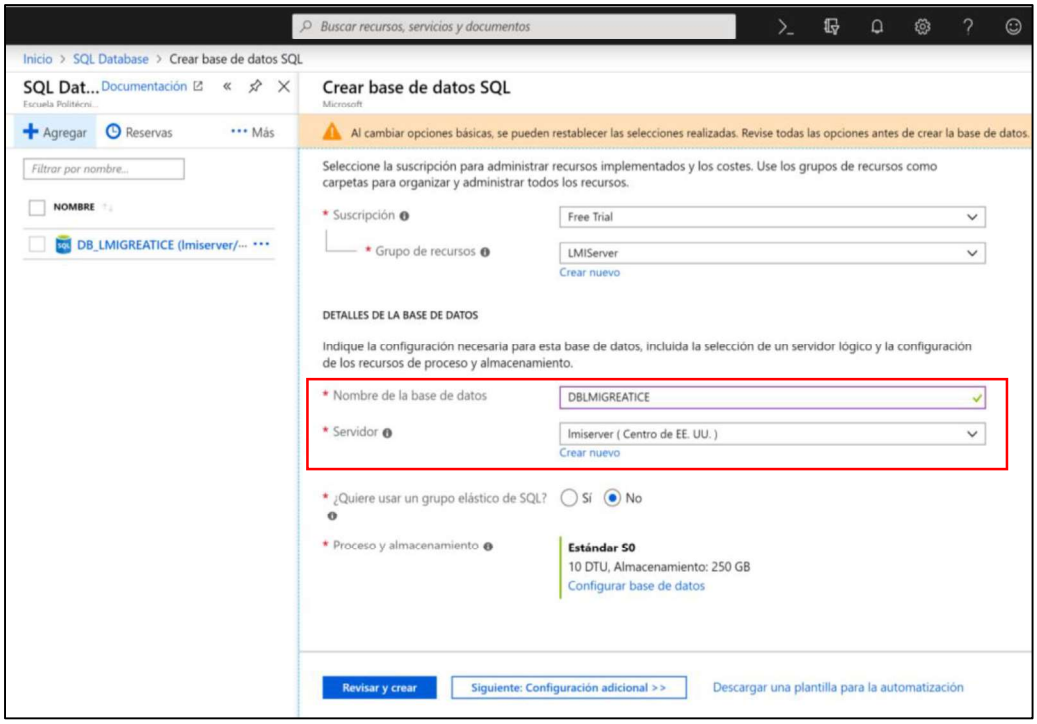


Figura 2.39 Creación de la base de datos en Azure

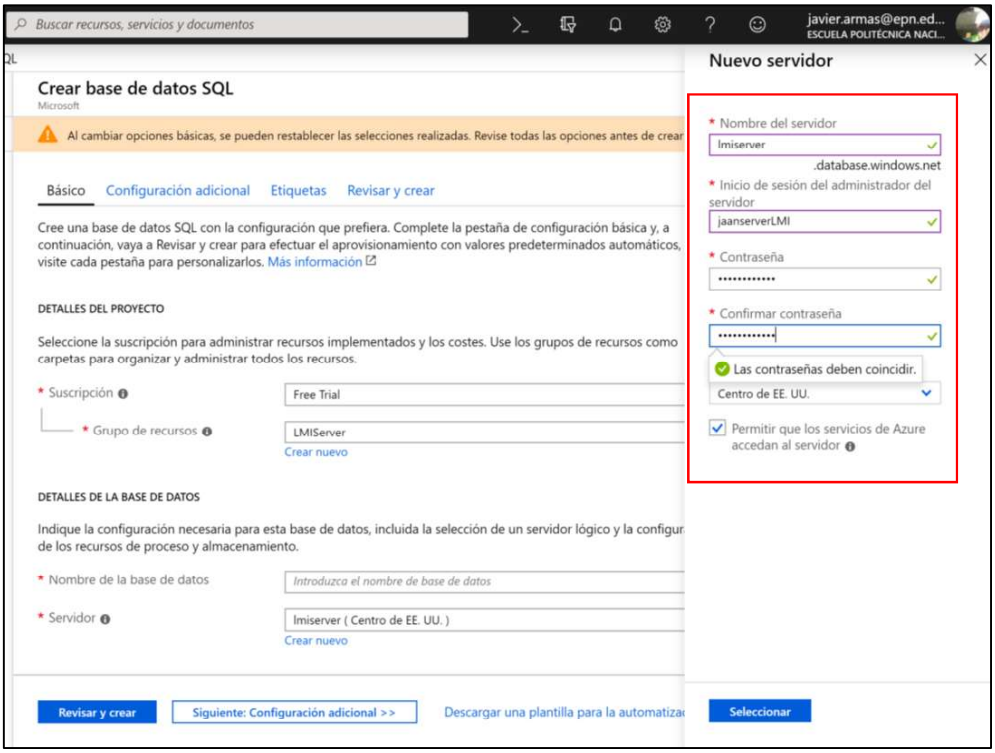


Figura 2.40 Creación de un nuevo servidor en Azure

Una vez generada la base de datos, es necesario conectarse al servidor creado utilizando las credenciales respectivas desde SQL Server Management Studio, tal como lo indica la Figura 2.41.

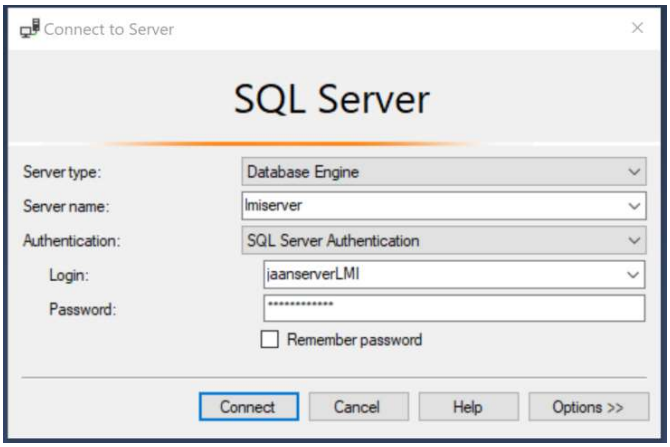


Figura 2.41 Conexión al servidor desde SQL Server Management Studio

3. RESULTADOS Y DISCUSIÓN

Se realizaron pruebas de funcionalidad para comprobar el correcto funcionamiento del sistema, tanto por módulos como en su totalidad.

Asimismo, se efectuó un conjunto de validaciones que permitieron constatar el cumplimiento de los requerimientos establecidos anteriormente.

Una vez que el sistema fue desplegado en Internet, se realizaron pruebas en los navegadores Google Chrome y Mozilla Firefox. Entre las tareas realizadas están:

- Validación de datos ingresados por el usuario, para constatar que la información que se ingresa sea la adecuada en cada Vista.
- Pruebas de autenticación de usuarios.
- Pruebas de búsqueda de información, que permitieron verificar si los resultados devueltos por el sistema al realizar una búsqueda corresponden a los que se solicitaron.
- Pruebas de creación de nuevos registros, ya sean de glaciares, montañas, usuarios, estaciones, países o instituciones.
- Pruebas de carga de nueva información al sistema, correspondiente a los documentos de meteorología, glaciología o hidrología.
- Validaciones de estructura de documentos, para verificar si se cumple con un formato establecido en función del tipo de medida, previo a la carga de información.
- Pruebas de modificación de registros de elementos como montañas, glaciares, usuarios, instituciones, países o estaciones.
- Pruebas de eliminación y desactivación de registros.
- Pruebas de descarga de información sobre los glaciares.

Por otra parte, se realizó una entrevista de cumplimiento de requerimientos que fue aplicada a 5 miembros del proyecto de investigación LMI GREAT ICE, para constatar su satisfacción con el sistema desarrollado.

Finalmente, se efectuó la corrección de los errores encontrados al probar el funcionamiento del sistema.

3.1 PRUEBAS DE FUNCIONAMIENTO

A continuación, se presentan algunos ejemplos de las pruebas realizadas.

3.1.1 CREACIÓN DE CUENTAS DE USUARIO

En la página principal del sistema, la opción `Register` permite crear nuevas cuentas de usuario. En la Vista de registro, mostrada en la Figura 3.1, el usuario debe ingresar la información personal solicitada. Es obligatorio llenar toda la información requerida. Adicionalmente, algunos campos necesitan que se cumpla con un formato establecido. Las validaciones se realizaron mediante *Data Annotations*, por lo que proveen instantáneamente un mensaje indicando el error al usuario, en caso de existir. El mensaje se muestra al cambiar de un control a otro.

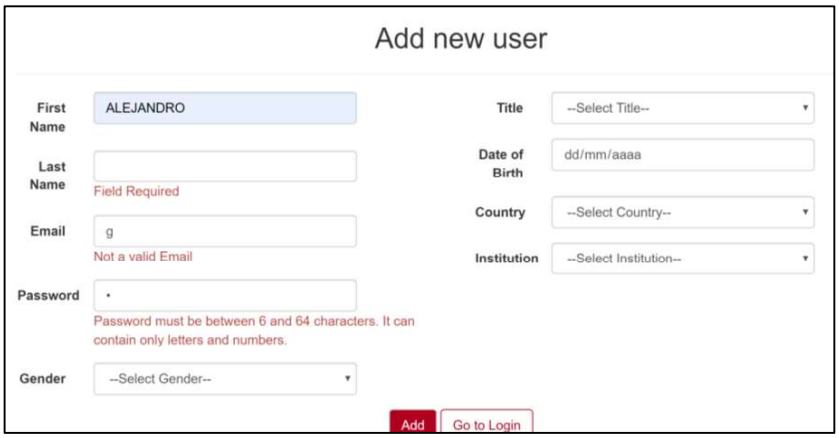


Figura 3.1 Mensajes de validación en la Vista de registro de usuarios

Si se intenta registrar un nuevo usuario con un correo electrónico que previamente ya fue registrado en el sistema, se visualizará una notificación indicándolo y la nueva cuenta no se guardará en la base de datos. Esto se puede visualizar en la Figura 3.2.

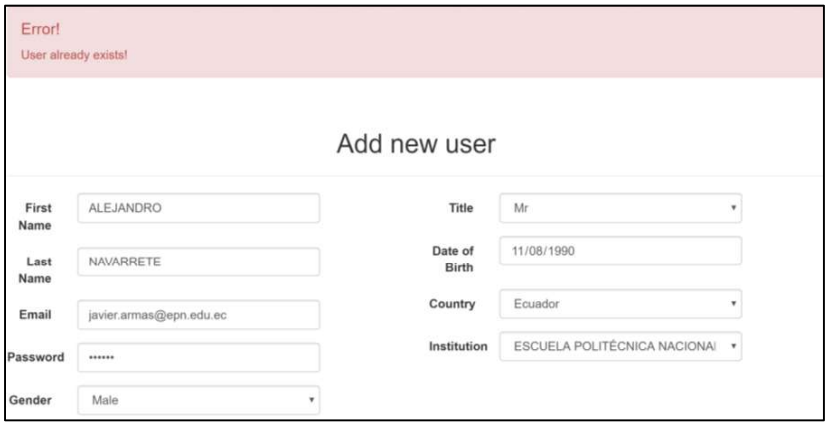


Figura 3.2 Notificación de error por existencia de usuario

En la Figura 3.3 se muestra un ejemplo correcto de registro de usuario, donde el correo electrónico que se desea registrar no existe en la base de datos y el resto de información solicitada cumple con los formatos establecidos.

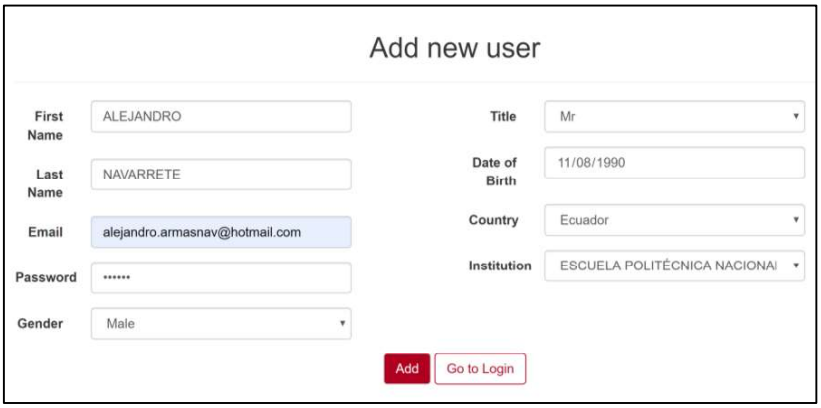


Figura 3.3 Ejemplo correcto de registro de un usuario

La información ingresada anteriormente se registra en la respectiva tabla de la base de datos, como se puede observar en la Figura 3.4.También, se puede verificar que la contraseña almacenada (columna `passwordU`) está cifrada.

	idSystemUser	firstName	lastName	email	passwordU	idGender	idTitle	dateOfBirth	idCountry
1	1	MARCOS	VILLACIS	marcos.villacis@epn.edu.ec	a332a7bf506a1e96682f41c4fa2987d772d77720e45d2dbe3...	2	3	1975-06-07	63
2	2	THOMAS	CONDOM	thomas.condom@ird.fr	a332a7bf506a1e96682f41c4fa2987d772d77720e45d2dbe3...	2	4	1983-09-04	74
3	3	ANTOINE	RABATEL	antoine.rabatel@univ-grenoble-...	855dcd67cf6cc60247b08c73fe74489d15022c2217504bc7...	2	2	1987-02-05	74
4	4	JAVIER	ARMAS	javier.armas@epn.edu.ec	a332a7bf506a1e96682f41c4fa2987d772d77720e45d2dbe3...	2	2	1990-08-11	63
5	45	ALEJANDRO	NAVARRETE	alejandro.armasnav@hotmail.com	a332a7bf506a1e96682f41c4fa2987d772d77720e45d2dbe3...	2	2	1990-08-11	63

Figura 3.4 Tabla `systemUser`

3.1.2 AUTENTICACIÓN DE USUARIOS

En la página principal del sistema, la opción `Login` redirige a una nueva Vista que permite al usuario iniciar sesión en el sistema con una cuenta previamente registrada. Para iniciar sesión, el usuario debe ingresar el correo electrónico y la contraseña asociada a su cuenta registrada. En caso de que uno de los dos campos ingresados sea incorrecto, se mostrará una notificación indicándolo. Esto se visualiza en la Figura 3.5.

Figura 3.5 Notificación de error por ingreso inválido de datos en la Vista `Login`

Si se ingresan adecuadamente el correo electrónico y la contraseña del usuario, se efectuará un inicio de sesión y, dependiendo del perfil del usuario que haya iniciado sesión, el sistema redirigirá a una u otra página.

En el caso de que el usuario tenga perfil `Download`, la página que se visualizará al iniciar sesión es la que se presenta en la Figura 3.6. En esta página se observan los datos de la cuenta del usuario, además, se tiene una opción que permite editar su información.

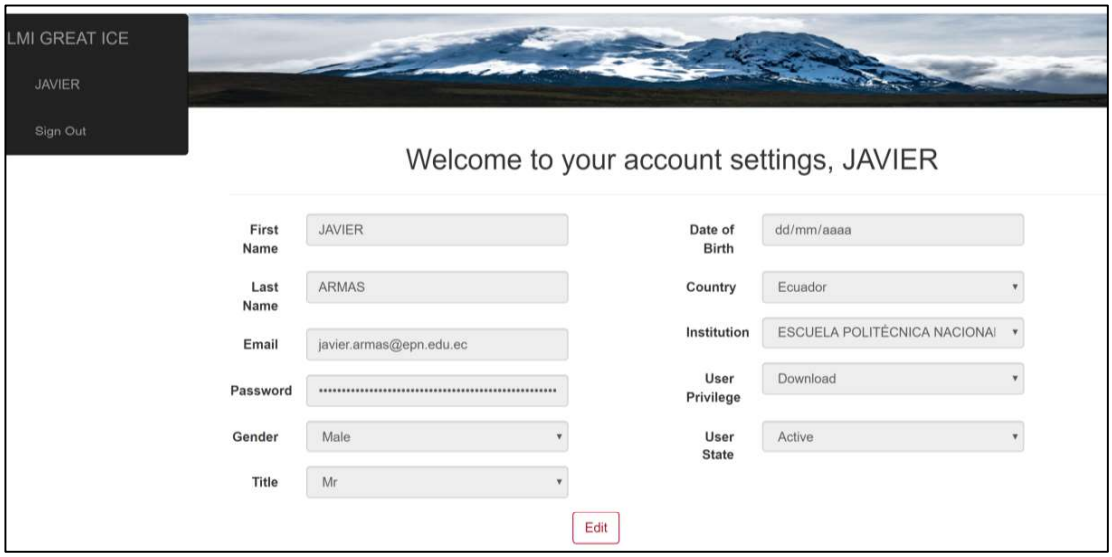


Figura 3.6 Inicio de sesión exitoso para un usuario con perfil `Download`

Si el usuario que ha iniciado sesión tiene perfil `Download&Upload`, la página que se mostrará es la indicada en la Figura 3.7. La diferencia entre esta página y la de un usuario

con perfil `Download`, es que tiene una opción en el panel de la izquierda que permite cargar datos al sistema.

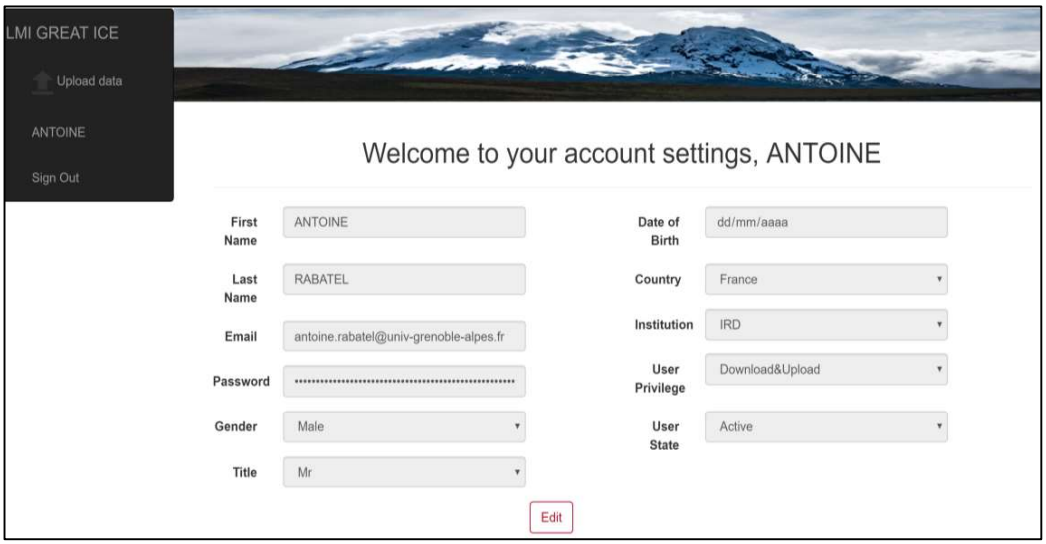


Figura 3.7 Inicio de sesión exitoso para un usuario con perfil `Download&Upload`

Finalmente, si el usuario que inició sesión tiene perfil `Administrator`, se mostrará la página de la Figura 3.8. Aquí se cuenta con un panel que permite la administración del sistema.

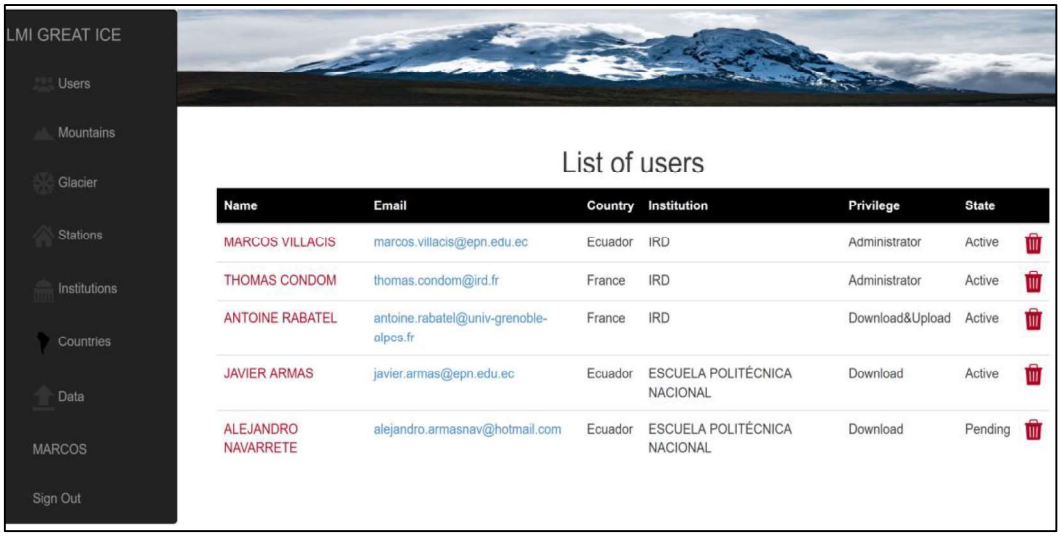


Figura 3.8 Inicio de sesión exitoso para un usuario con perfil `Administrator`

En caso de que el usuario haya olvidado la contraseña con la que creó su cuenta, existe una opción que le permitirá restablecerla. Esta opción se encuentra en la página de `Login`. Al seleccionar esta opción, el sistema redirigirá a una nueva página que solicitará el ingreso del correo electrónico del usuario, como se observa en la Figura 3.9. Se enviará una notificación a ese correo, conjuntamente con una contraseña temporal para que el usuario pueda ingresar al sistema y cambiarla posteriormente por una propia, como se muestra en la Figura 3.10.

Figura 3.9 Página para el restablecimiento de contraseña

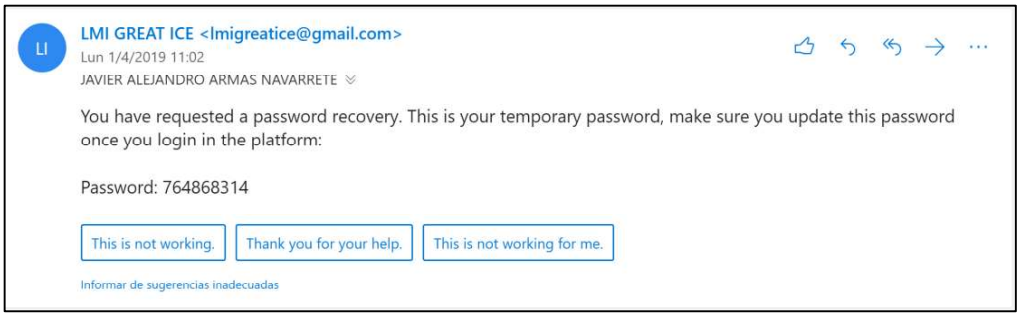


Figura 3.10 Correo electrónico enviado al solicitar restablecimiento de contraseña

3.1.3 BÚSQUEDA DE INFORMACIÓN

El sistema desarrollado permite realizar búsquedas de información sobre glaciares mediante la selección de distintas opciones para realizar un filtrado de datos.

Las opciones a seleccionar son: País → Montaña → Glaciar → Tipo de dato → Tipo de medida → Estación → Variable → Año (Si el tipo de medida es Meteorología o Hidrología), o Documento (Si el tipo de medida es Glaciología) → Franja altitudinal (Si el tipo de medida es Glaciología).

La búsqueda de información se la puede realizar desde la página principal del sistema, como se muestra en la Figura 3.11.

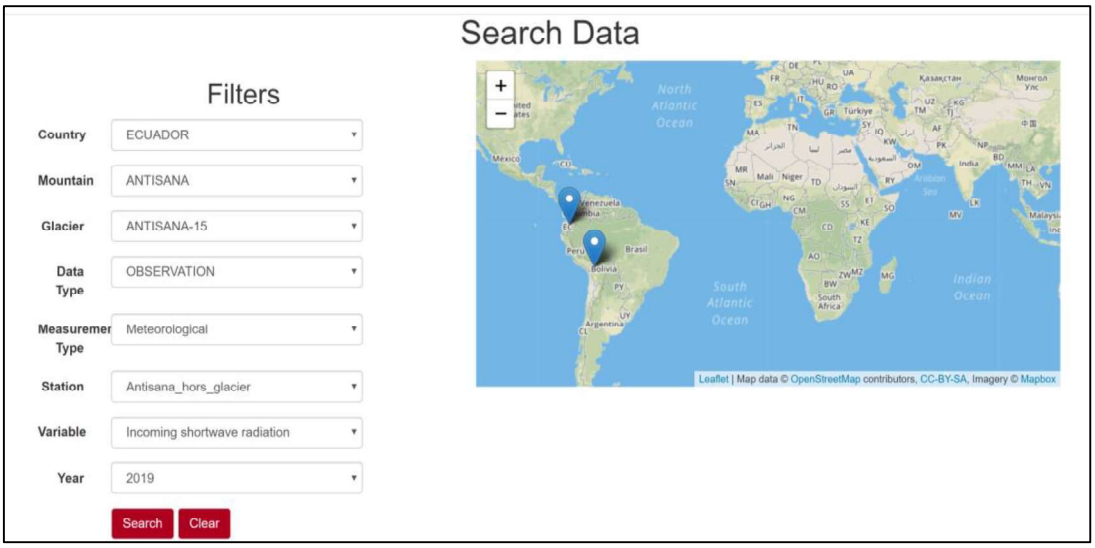


Figura 3.11 Búsqueda de información desde la página principal del sistema

Los resultados de la búsqueda se despliegan en forma de una gráfica temporal. Adicionalmente se muestra información relativa a la estación que recolectó los datos. Todo esto se presenta en una nueva Vista, como se observa en la Figura 3.12.

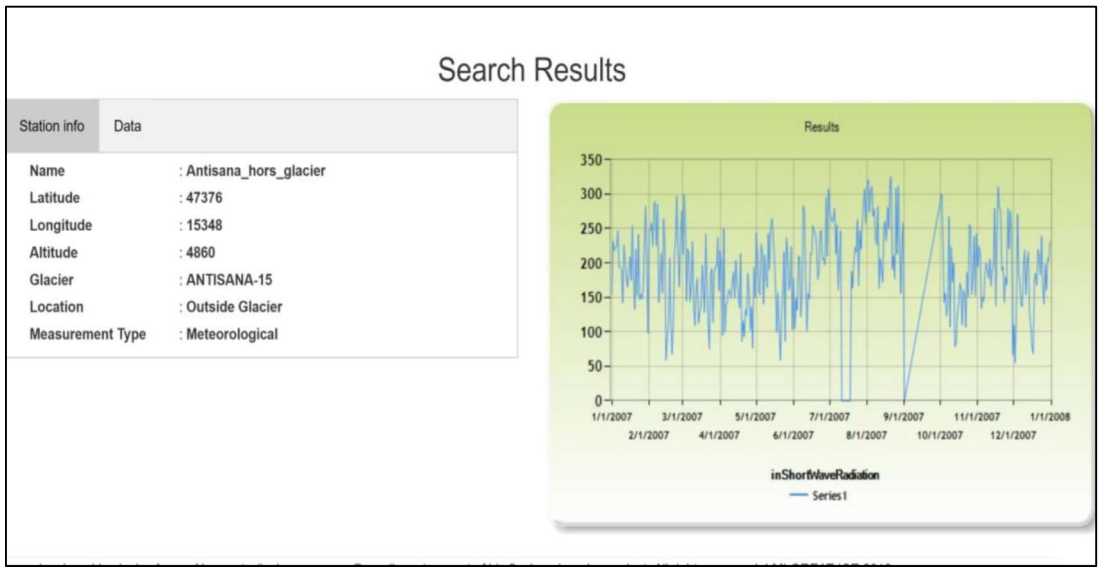


Figura 3.12 Vista de Resultados

Un usuario que ha iniciado sesión puede descargar los resultantes correspondientes a la búsqueda que se realizó. El archivo descargado tiene formato CSV (*comma-separated-values*). Esto se puede observar en la Figura 3.13 y en la Figura 3.14.

Figura 3.13 Archivo descargado con los resultados obtenidos en formato CSV

Figura 3.14 Contenido del archivo CSV descargado

3.1.4 CREACIÓN DE NUEVOS REGISTROS

Un usuario con perfil `Administrator` puede agregar nuevos registros de glaciares, montañas, instituciones, estaciones, países o incluso usuarios.

En la Figura 3.15 se muestra la página que permite agregar nuevas estaciones. Todos los campos deben llenarse de forma obligatoria.

Si los campos ingresados son válidos y previamente no se ha agregado una estación con el mismo nombre, la información especificada se guardará como un nuevo registro en la tabla correspondiente de la base de datos. Esto se visualiza en la Figura 3.16.

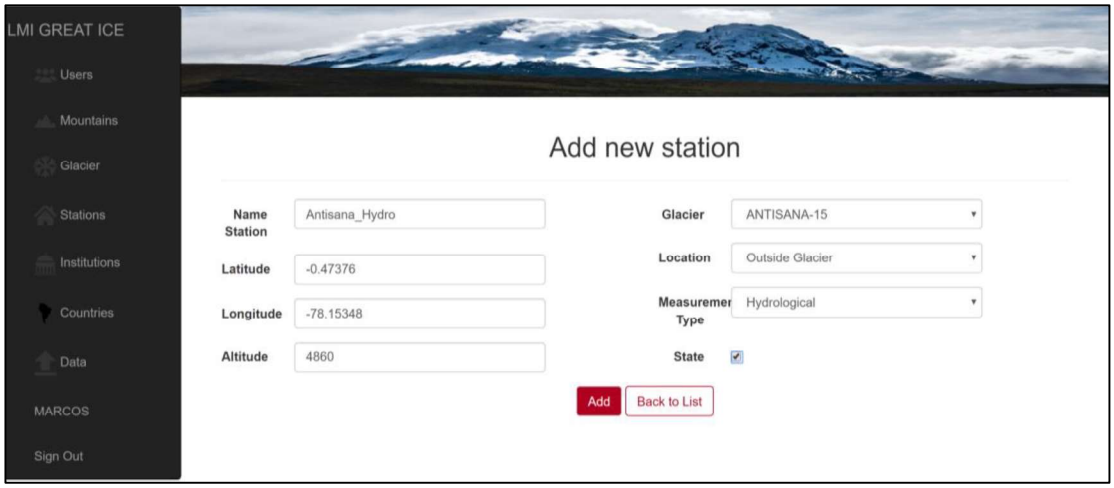


Figura 3.15 Página para agregar nuevas estaciones

	idStation	nameStation	idLocation	latitudeStation	longitudeStation	altitudeStation	idGlacier	idMeasurementType	stateR
1	1	Antisana_hors_glacier	2	-0.47376	-78.15348	4860	1	1	1
2	2	Antisana_sur_glacier	1	-0.47481	-78.15337	4890	1	1	1
3	3	Zongo_hors_glacier	2	-16.28234	-68.13678	5050	2	1	1
4	4	Zongo_sur_glacier	1	-16.27983	-68.14072	5035	2	1	1
5	5	Antisana_Mass_Balance	1	-0.47	-78.15	5025	1	2	1
6	6	Zongo_Mass_Balance	1	-16.27	-68.14	5150	2	2	1
7	8	Antisana_Hydro	2	-0.47376	-78.15348	4860	1	3	1

Figura 3.16 Tabla `stations`

3.1.5 MODIFICACIÓN DE REGISTROS

Los usuarios con perfil `Administrator` pueden realizar la modificación de registros de montañas, glaciares, estaciones, entre otros.

La Figura 3.17 muestra la página correspondiente a la edición de registros de glaciares. Se ha seleccionado el glaciar `ANTISANA-15` para ser modificado.

En la Figura 3.18 se muestra la tabla `glaciers` con el registro del glaciar `ANTISANA-15` antes de ser modificado.

Únicamente se realizará un cambio al nombre del glaciar. De esta manera el nuevo nombre será Antisana-12, como se muestra en la Figura 3.19.

El cambio en el nombre del registro se ve reflejado en la tabla correspondiente de la base de datos, tal como se puede visualizar en la Figura 3.20.

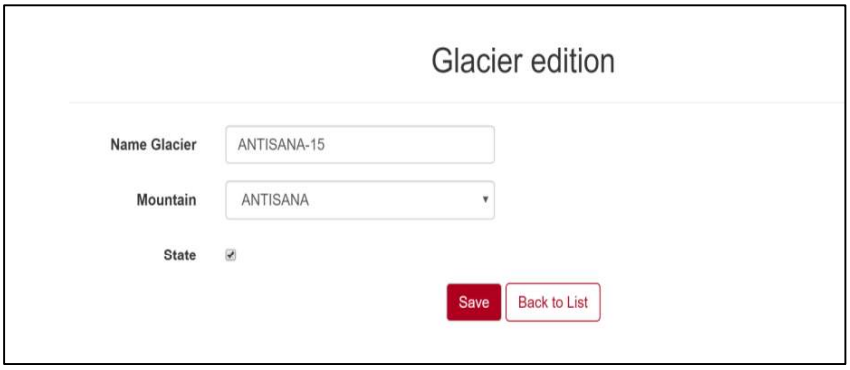


Figura 3.17 Página para edición de registros de glaciares

	idGlacier	nameGlacier	idMountain	stateR
1	1	ANTISANA-15	1	1
2	2	ZONGO	2	1

Figura 3.18 Tabla `glaciers` antes de modificar un registro

Figura 3.19 Modificación de registros de glaciares

	idGlacier	nameGlacier	idMountain	stateR
1	1	ANTISANA-12	1	1
2	2	ZONGO	2	1

Figura 3.20 Tabla `glaciers` después de modificar un registro

3.1.6 ELIMINACIÓN Y DESACTIVACIÓN DE REGISTROS

Existen registros que por cuestiones de consistencia no deben ser eliminados de la base de datos. Sin embargo, pueden desactivarse para que ya no se muestren al usuario, manteniendo su presencia en la tabla respectiva de la base de datos. Ejemplos de registros que no deben eliminarse son los correspondientes a montañas, glaciares, estaciones o usuarios. Esto se debe a que están relacionados estrechamente y al eliminar uno de ellos, causaría inconsistencias en el sistema.

En la Figura 3.21 se muestra la lista de glaciares que están guardados en la base de datos. Como se puede observar, los casilleros de la columna `State` están seleccionados, lo que implica que los registros de ambos glaciares están activados.

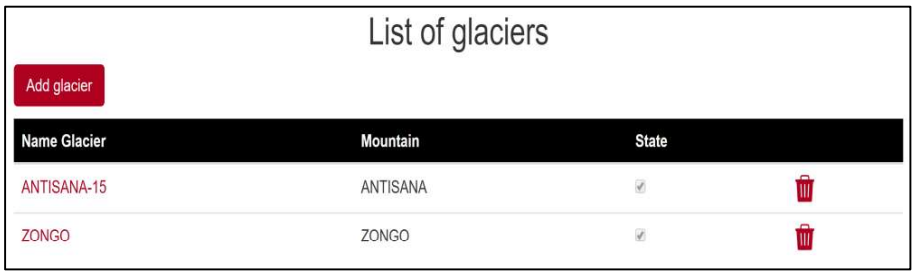


Figura 3.21 Lista de glaciares guardados en la base de datos

La lista de glaciares es utilizada por ejemplo, al crear una nueva estación, donde se debe seleccionar el glaciar en el que se encuentra ubicada. En la Figura 3.22 se despliegan los glaciares cuyos registros están activados, en este caso `ANTISANA-15 y ZONGO`.

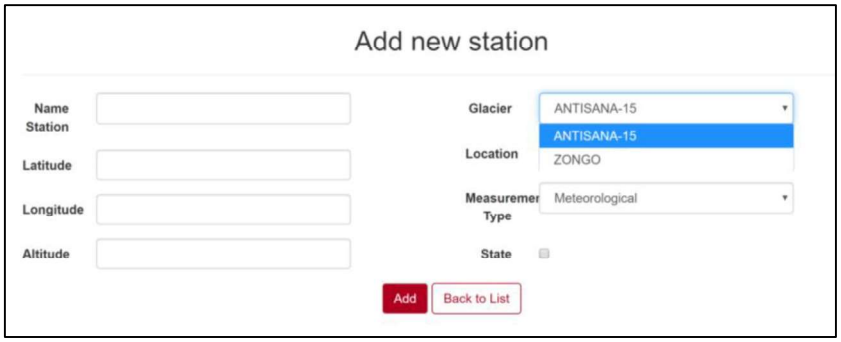


Figura 3.22 Despliegue de los registros activados de glaciares

Al desactivar un registro, la columna `State` del registro desactivado permanece sin selección, como se observa en la Figura 3.23.

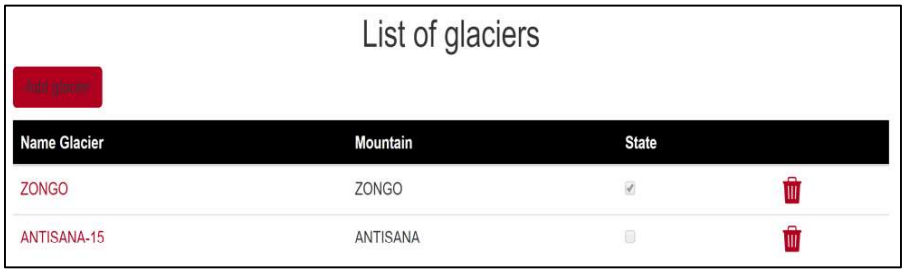


Figura 3.23 Desactivación de registros de glaciares

Los registros desactivados no se muestran al usuario. Esto se puede verificar en la Figura 3.24, donde al desplegar la lista de glaciares, no aparece `ANTISANA-15`, puesto que previamente el registro fue desactivado.

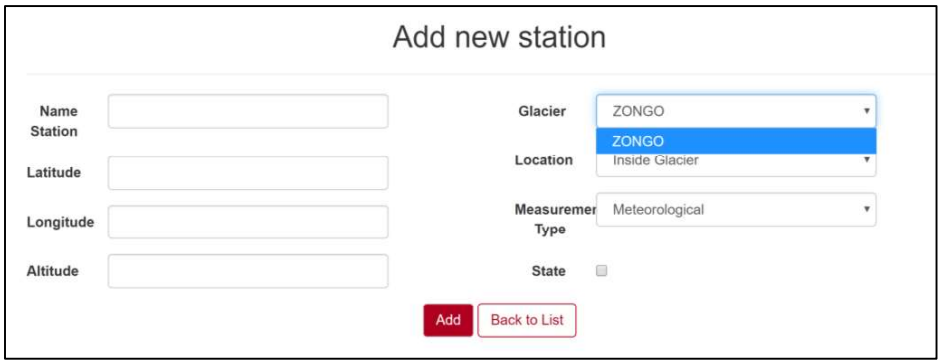


Figura 3.24 Despliegue de los glaciares activados. `ANTISANA-15` no desplegado

Por otra parte, hay otros registros que si pueden y deben eliminarse de la base de datos. Estos corresponden a los registros de los documentos que se suben con información meteorológica, glaciológica o hidrológica. La razón para que se puedan eliminar es que un usuario puede desear subir de nuevo un documento que previamente ha sido subido pero que tenía errores. Dado que no se permite la modificación en línea, se deben eliminar todos los registros correspondientes a ese documento y luego subirlo nuevamente. En la Figura 3.25 se muestra la página donde se listan los registros de documentos y desde donde se

puede eliminarlos, esto se logra al seleccionar el ícono de basurero que se encuentra en la parte derecha de cada registro.

Figura 3.25 Página para eliminar registros de documentos

En la Figura 3.26 se muestra la tabla `enteredData` antes de eliminar cualquier registro. Se puede observar que el último registro de documento corresponde a aquel que tiene el campo `idEnteredData` igual a 66, y el nombre del documento es `B_monthly_ABL_zongo_2013`.

	idEnteredData	nameEnteredData	startDate	endDate	idDataType	idStation	idSystemUser	stateR
32	32	B_monthly_ABL_zongo_2004.csv	2004-01-01 00:00:00.000	2005-01-01 00:00:00.000	1	6	3	1
33	33	B_monthly_ABL_zongo_2005.csv	2005-01-01 00:00:00.000	2006-01-01 00:00:00.000	1	6	3	1
34	34	B_monthly_ABL_zongo_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	6	3	1
35	35	B_monthly_ABL_zongo_2007.csv	2007-01-01 00:00:00.000	2007-12-29 00:00:00.000	1	6	3	1
36	36	B_monthly_ABL_zongo_2008.csv	2007-12-29 00:00:00.000	2009-01-02 00:00:00.000	1	6	3	1
37	37	B_monthly_ABL_zongo_2009.csv	2009-01-02 00:00:00.000	2010-01-02 00:00:00.000	1	6	3	1
38	38	B_monthly_ABL_zongo_2010.csv	2010-01-02 00:00:00.000	2011-01-03 00:00:00.000	1	6	3	1
39	39	B_monthly_ABL_zongo_2011.csv	2011-01-03 00:00:00.000	2012-01-05 00:00:00.000	1	6	3	1
40	40	B_monthly_ABL_zongo_2012.csv	2012-01-05 00:00:00.000	2013-01-05 00:00:00.000	1	6	3	1
41	43	Antisana_HorsGlacier_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	1	1	1
42	44	Antisana_HorsGlacier_2007.csv	2007-01-01 00:00:00.000	2008-01-01 00:00:00.000	1	1	1	1
43	55	Antisana_HorsGlacier_2008.csv	2008-01-01 00:00:00.000	2009-01-01 00:00:00.000	1	1	1	1
44	66	B_monthly_ABL_zongo_2013.csv	2013-01-05 00:00:00.000	2013-05-29 00:00:00.000	1	5	3	1

Figura 3.26 Tabla `enteredData` antes de la eliminación de registros

La Figura 3.27 corresponde a la Tabla `glaciologicalData` antes de la eliminación de registros. En esta tabla se almacenan los datos glaciológicos asociados a los documentos.

Se puede visualizar que existen registros asociados al documento cuyo `idEnteredData` es igual a 66.

Una vez que se elimina el registro de un documento, también se eliminan los registros asociados a dicho documento.

Por ejemplo, al eliminar el documento cuyo `idEnteredData` es igual a 66, su registro se elimina de la tabla `enteredData` de la base de datos. Esto se observa en la Figura 3.28, donde se visualiza que ya no existe un registro de documento con el `idEnteredData` indicado.

Adicionalmente, también se eliminan los registros asociados al documento eliminado, como se puede constatar en la Figura 3.29 correspondiente a la tabla `glaciologicalData`, en donde no existen registros de glaciología asociados al documento con `idEnteredData` igual a 66.

	idMassBalanceData	altitudinalBar	startDay	startMonth	startYear	endDay	endMonth	endYear	latitude	longitude	altitude	massBalance	monthData	idEnteredDat
1...	1964	5200_5300	29	2	2013	5	4	2013	-16.279...	-68.146857	5250	0.01	3	66
1...	1965	5100_5200	29	2	2013	5	4	2013	-16.277...	-68.143986	5150	-0.09	3	66
1...	1966	5000_5100	29	2	2013	5	4	2013	-16.279...	-68.139232	5050	-0.29	3	66
1...	1967	4900_5000	29	2	2013	5	4	2013	-16.279...	-68.135147	4950	-0.96	3	66
1...	1968	5200_5300	5	4	2013	19	4	2013	-16.279...	-68.146857	5250	-0.15	4	66
1...	1969	5100_5200	5	4	2013	19	4	2013	-16.277...	-68.143986	5150	-0.1	4	66
1...	1970	5000_5100	5	4	2013	19	4	2013	-16.279...	-68.139232	5050	-0.43	4	66
1...	1971	4900_5000	5	4	2013	19	4	2013	-16.279...	-68.135147	4950	-6999	4	66
1...	1972	5200_5300	19	4	2013	29	5	2013	-16.279...	-68.146857	5250	0.33	5	66
1...	1973	5100_5200	19	4	2013	29	5	2013	-16.277...	-68.143986	5150	0.09	5	66
1...	1974	5000_5100	19	4	2013	29	5	2013	-16.279...	-68.139232	5050	-0.1	5	66
1...	1975	4900_5000	19	4	2013	29	5	2013	-16.279...	-68.135147	4950	-0.26	5	66

Figura 3.27 Tabla `glaciologicalData` antes de la eliminación de registros

	idEnteredData	nameEnteredData	startDate	endDate	idDataType	idStation	idSystemUser	stateR
31	31	B_monthly_ABL_zongo_2003.csv	2003-01-01 00:00:00.000	2004-01-01 00:00:00.000	1	6	3	1
32	32	B_monthly_ABL_zongo_2004.csv	2004-01-01 00:00:00.000	2005-01-01 00:00:00.000	1	6	3	1
33	33	B_monthly_ABL_zongo_2005.csv	2005-01-01 00:00:00.000	2006-01-01 00:00:00.000	1	6	3	1
34	34	B_monthly_ABL_zongo_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	6	3	1
35	35	B_monthly_ABL_zongo_2007.csv	2007-01-01 00:00:00.000	2007-12-29 00:00:00.000	1	6	3	1
36	36	B_monthly_ABL_zongo_2008.csv	2007-12-29 00:00:00.000	2009-01-02 00:00:00.000	1	6	3	1
37	37	B_monthly_ABL_zongo_2009.csv	2009-01-02 00:00:00.000	2010-01-02 00:00:00.000	1	6	3	1
38	38	B_monthly_ABL_zongo_2010.csv	2010-01-02 00:00:00.000	2011-01-03 00:00:00.000	1	6	3	1
39	39	B_monthly_ABL_zongo_2011.csv	2011-01-03 00:00:00.000	2012-01-05 00:00:00.000	1	6	3	1
40	40	B_monthly_ABL_zongo_2012.csv	2012-01-05 00:00:00.000	2013-01-05 00:00:00.000	1	6	3	1
41	43	Antisana_HorsGlacier_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	1	1	1
42	44	Antisana_HorsGlacier_2007.csv	2007-01-01 00:00:00.000	2008-01-01 00:00:00.000	1	1	1	1
43	55	Antisana_HorsGlacier_2008.csv	2008-01-01 00:00:00.000	2009-01-01 00:00:00.000	1	1	1	1

Figura 3.28 Tabla `enteredData` después de la eliminación de registros

	idMassBalanceData	altitudinalBar	startDay	startMonth	startYear	endDay	endMonth	endYear	latitude	longitude	altitude	massBalance	monthData	idEnteredDat
1...	1744	5200_5300	30	9	2011	27	10	2012	-16.279...	-68.146857	5250	-0.35	10	40
1...	1745	5100_5200	30	9	2011	27	10	2012	-16.277...	-68.143986	5150	-0.48	10	40
1...	1746	5000_5100	30	9	2011	27	10	2012	-16.279...	-68.139232	5050	-0.78	10	40
1...	1747	4900_5000	30	9	2011	27	10	2012	-16.279...	-68.135147	4950	-1.14	10	40
1...	1748	5200_5300	27	10	2011	1	12	2012	-16.279...	-68.146857	5250	-0.19	11	40
1...	1749	5100_5200	27	10	2011	1	12	2012	-16.277...	-68.143986	5150	-0.33	11	40
1...	1750	5000_5100	27	10	2011	1	12	2012	-16.279...	-68.139232	5050	-0.53	11	40
1...	1751	4900_5000	27	10	2011	1	12	2012	-16.279...	-68.135147	4950	-0.53	11	40
1...	1752	5200_5300	1	12	2012	5	1	2013	-16.279...	-68.146857	5250	0.06	12	40
1...	1753	5100_5200	1	12	2012	5	1	2013	-16.277...	-68.143986	5150	0.01	12	40
1...	1754	5000_5100	1	12	2012	5	1	2013	-16.279...	-68.139232	5050	-0.2	12	40
1...	1755	4900_5000	1	12	2012	5	1	2013	-16.279...	-68.135147	4950	-0.58	12	40

Figura 3.29 Tabla `glaciologicalData` después de la eliminación de registros

3.1.7 CARGA DE NUEVOS DOCUMENTOS AL SISTEMA

El sistema desarrollado permite cargar nuevos documentos con información meteorológica, glaciológica o hidrológica. La información es almacenada en las respectivas tablas de la base de datos.

En la Figura 3.30 se muestra la página que permite cargar nuevos documentos.

Todos los campos solicitados son obligatorios. En este caso, se va a cargar el documento que previamente fue eliminado, `B_monthly_ABL_zongo_2013.csv`.

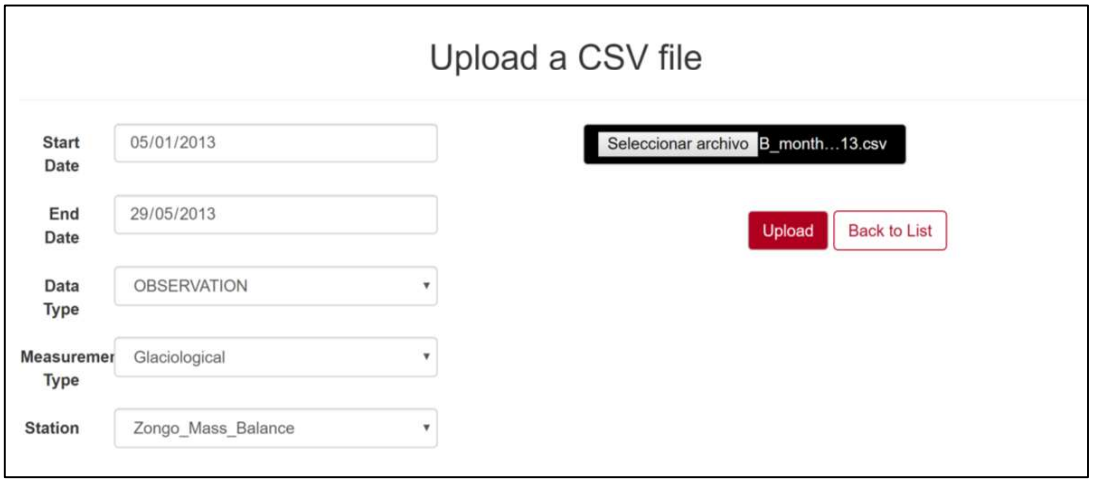


Figura 3.30 Ejemplo de carga de nuevos documentos

En la Figura 3.31 se muestra la tabla `enteredData`. Puede observarse que el último registro de esta tabla tiene un `idEnteredData` igual a 55, correspondiente al documento

`Antisana_HorsGlacier_2008.csv`. No existe registro del documento que se desea subir.

	idEnteredData	nameEnteredData	startDate	endDate	idDataType	idStation	idSystemUser	stateR
31	31	B_monthly_ABL_zongo_2003.csv	2003-01-01 00:00:00.000	2004-01-01 00:00:00.000	1	6	3	1
32	32	B_monthly_ABL_zongo_2004.csv	2004-01-01 00:00:00.000	2005-01-01 00:00:00.000	1	6	3	1
33	33	B_monthly_ABL_zongo_2005.csv	2005-01-01 00:00:00.000	2006-01-01 00:00:00.000	1	6	3	1
34	34	B_monthly_ABL_zongo_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	6	3	1
35	35	B_monthly_ABL_zongo_2007.csv	2007-01-01 00:00:00.000	2007-12-29 00:00:00.000	1	6	3	1
36	36	B_monthly_ABL_zongo_2008.csv	2007-12-29 00:00:00.000	2009-01-02 00:00:00.000	1	6	3	1
37	37	B_monthly_ABL_zongo_2009.csv	2009-01-02 00:00:00.000	2010-01-02 00:00:00.000	1	6	3	1
38	38	B_monthly_ABL_zongo_2010.csv	2010-01-02 00:00:00.000	2011-01-03 00:00:00.000	1	6	3	1
39	39	B_monthly_ABL_zongo_2011.csv	2011-01-03 00:00:00.000	2012-01-05 00:00:00.000	1	6	3	1
40	40	B_monthly_ABL_zongo_2012.csv	2012-01-05 00:00:00.000	2013-01-05 00:00:00.000	1	6	3	1
41	43	Antisana_HorsGlacier_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	1	1	1
42	44	Antisana_HorsGlacier_2007.csv	2007-01-01 00:00:00.000	2008-01-01 00:00:00.000	1	1	1	1
43	55	Antisana_HorsGlacier_2008.csv	2008-01-01 00:00:00.000	2009-01-01 00:00:00.000	1	1	1	1

Figura 3.31 Tabla `enteredData` antes de cargar un nuevo documento

Una vez que se sube el documento deseado el registro del documento se ve reflejado en la tabla `enteredData`. Mientras que, por tratarse de un documento de glaciología, la información contenida en el mismo se verá reflejada en la tabla `glaciologicalData`. Esto se puede observar en la Figura 3.32 y en la Figura 3.33. En la primera se visualiza el registro creado para el documento `B_monthly_ABL_zongo_2013.csv` con un `idEnteredData` igual a 67; en la segunda, se demuestra que la información contenida en el documento ha sido cargada en la tabla `glaciologicalData` con su `idEnteredData` respectivo.

	idEnteredData	nameEnteredData	startDate	endDate	idDataType	idStation	idSystemUser	stateR
32	32	B_monthly_ABL_zongo_2004.csv	2004-01-01 00:00:00.000	2005-01-01 00:00:00.000	1	6	3	1
33	33	B_monthly_ABL_zongo_2005.csv	2005-01-01 00:00:00.000	2006-01-01 00:00:00.000	1	6	3	1
34	34	B_monthly_ABL_zongo_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	6	3	1
35	35	B_monthly_ABL_zongo_2007.csv	2007-01-01 00:00:00.000	2007-12-29 00:00:00.000	1	6	3	1
36	36	B_monthly_ABL_zongo_2008.csv	2007-12-29 00:00:00.000	2009-01-02 00:00:00.000	1	6	3	1
37	37	B_monthly_ABL_zongo_2009.csv	2009-01-02 00:00:00.000	2010-01-02 00:00:00.000	1	6	3	1
38	38	B_monthly_ABL_zongo_2010.csv	2010-01-02 00:00:00.000	2011-01-03 00:00:00.000	1	6	3	1
39	39	B_monthly_ABL_zongo_2011.csv	2011-01-03 00:00:00.000	2012-01-05 00:00:00.000	1	6	3	1
40	40	B_monthly_ABL_zongo_2012.csv	2012-01-05 00:00:00.000	2013-01-05 00:00:00.000	1	6	3	1
41	43	Antisana_HorsGlacier_2006.csv	2006-01-01 00:00:00.000	2007-01-01 00:00:00.000	1	1	1	1
42	44	Antisana_HorsGlacier_2007.csv	2007-01-01 00:00:00.000	2008-01-01 00:00:00.000	1	1	1	1
43	55	Antisana_HorsGlacier_2008.csv	2008-01-01 00:00:00.000	2009-01-01 00:00:00.000	1	1	1	1
44	67	B_monthly_ABL_zongo_2013.csv	2013-01-05 00:00:00.000	2013-05-29 00:00:00.000	1	6	1	1

Figura 3.32 Tabla `enteredData` luego de cargar el documento deseado

	idMassBalanceData	altitudinalBar	startDay	startMonth	startYear	endDay	endMonth	endYear	latitude	longitude	altitude	massBalance	monthData	idEnteredDat
1...	1984	5200_5300	29	2	2013	5	4	2013	-16.279...	-68.146857	5250	0.01	3	67
1...	1985	5100_5200	29	2	2013	5	4	2013	-16.277...	-68.143986	5150	-0.09	3	67
1...	1986	5000_5100	29	2	2013	5	4	2013	-16.279...	-68.139232	5050	-0.29	3	67
1...	1987	4900_5000	29	2	2013	5	4	2013	-16.279...	-68.135147	4950	-0.96	3	67
1...	1988	5200_5300	5	4	2013	19	4	2013	-16.279...	-68.146857	5250	-0.15	4	67
1...	1989	5100_5200	5	4	2013	19	4	2013	-16.277...	-68.143986	5150	-0.1	4	67
1...	1990	5000_5100	5	4	2013	19	4	2013	-16.279...	-68.139232	5050	-0.43	4	67
1...	1991	4900_5000	5	4	2013	19	4	2013	-16.279...	-68.135147	4950	-6999	4	67
1...	1992	5200_5300	19	4	2013	29	5	2013	-16.279...	-68.146857	5250	0.33	5	67
1...	1993	5100_5200	19	4	2013	29	5	2013	-16.277...	-68.143986	5150	0.09	5	67
1...	1994	5000_5100	19	4	2013	29	5	2013	-16.279...	-68.139232	5050	-0.1	5	67
1...	1995	4900_5000	19	4	2013	29	5	2013	-16.279...	-68.135147	4950	-0.26	5	67

Figura 3.33 Tabla `glaciologicalData`

3.2 PRUEBAS DE VALIDACIÓN

Se solicitó a los miembros del proyecto de investigación LMI GREAT ICE utilizar el sistema desarrollado para la realización de las pruebas respectivas. Considerando que el sistema admite 3 perfiles de usuario con distintos privilegios, se elaboraron 3 formatos distintos de entrevistas. En total, se aplicaron 5 entrevistas a los usuarios con perfil `Administrator`, `Download&Upload` y `Download`. Los formatos de entrevistas aplicados y sus resultados se encuentran en el ANEXO E.

En la Tabla 3.1 y la Tabla 3.2 se muestra un resumen de los resultados de las entrevistas aplicadas a usuarios con perfil `Download`.

En la Tabla 3.3 y la Tabla 3.4 se muestran los resultados de las entrevistas aplicadas a usuarios con perfil `Download&Upload`.

Finalmente, la Tabla 3.5, la Tabla 3.6, la Tabla 3.7 y la Tabla 3.8 presentan los resultados de las entrevistas aplicadas a usuarios con perfil `Administrator`.

Tabla 3.1 Resultados de entrevistas - perfil `Download` (parte 1 de 2)

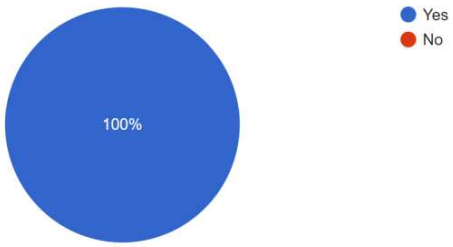
No	Pregunta	Resultados
1	Please log in at the system considering the following credentials: Email=javier.armas@epn.edu.ec / Password: LMI123 Could you do it?	

Tabla 3.2 Resultados de entrevistas - perfil `Download` (parte 2 de 2)

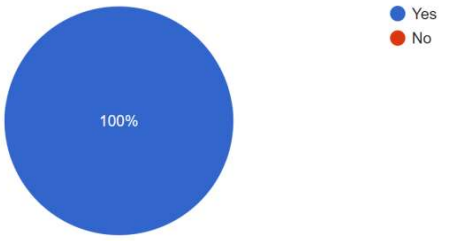
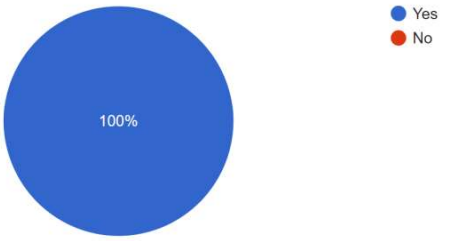
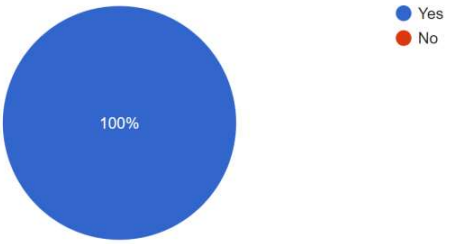
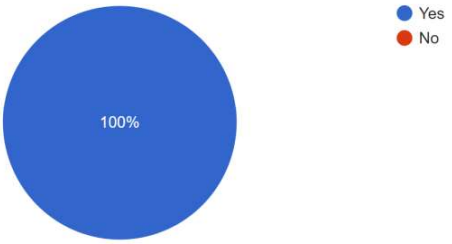
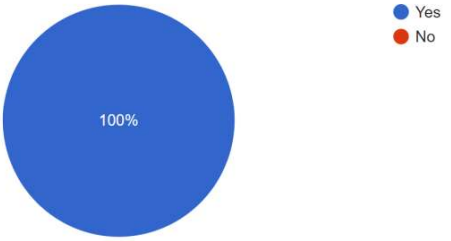
No	Pregunta	Resultados
2	Now, select the option Search Data and look for information considering the following parameters: Country=Ecuador / Mountain=Antisana / Glacier=Antisana-15 / Data Type=Observation / Measurement Type=Meteorological / Station=Antisana_hors_glacier / Variable=Incoming short wave radiation / Year=2007. Could you do it and visualize a temporary graph with the results?	
3	Could you also visualize information associated with the station selected?	
4	Please try to download the results. Could you do it?	
5	Now, go to the main page by clicking "LMI GREAT ICE" text at the bottom of the page. Then, in the map, select the marker called Zongo_Mass_Balance. Each marker corresponds to a station of LMI GREAT ICE. Could you visualize the documents associated with the selected station?	
6	Please create a new user account. Could you create it? Note: You will not be able to log in at the system with your new user account because it must requires the system administrator to activate it.	

Tabla 3.3 Resultados de entrevistas - perfil `Download&Upload` (parte 1 de 2)

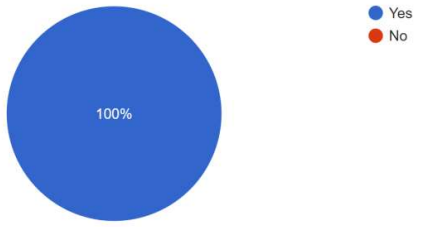
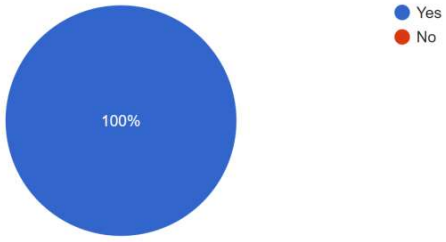
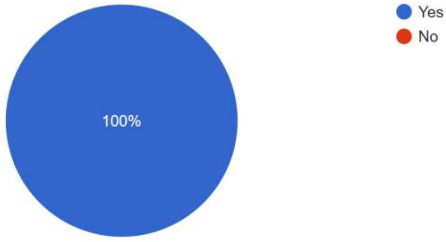
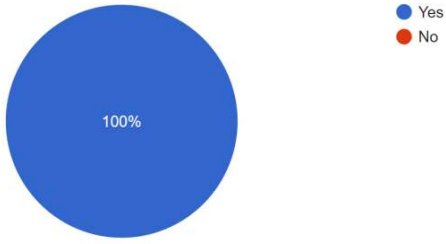
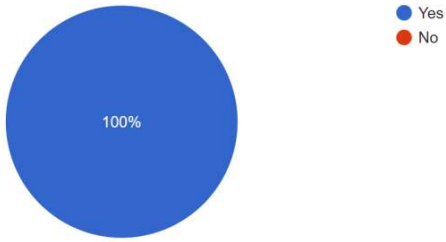
No	Pregunta	Resultados
1	Please log in at the system considering the following credentials: Email=antoine.rabatel@univ-grenoble-alpes.fr / Password: 78LMI123456 Could you do it?	
2	Now, select the option "Search Data" and look for information considering the following parameters: Country=Ecuador / Mountain=Antisana / Glacier=Antisana-15 / Data Type=Observation / Measurement Type=Meteorological / Station=Antisana_hors_glacier / Variable=Incoming short wave radiation / Year=2007. Could you do it and visualize a temporary graph with the results?	
3	Could you also visualize information associated with the station selected?	
4	Please try to download the results. Could you do it?	
5	Now, go to the main page by clicking "LMI GREAT ICE" text at the bottom of the page. Then, in the map, select the marker called Zongo_Mass_Balance. Each marker corresponds to a station of LMI GREAT ICE. Could you visualize the documents associated with the selected station?	

Tabla 3.4 Resultados de entrevistas - perfil `Download&Upload` (parte 2 de 2)

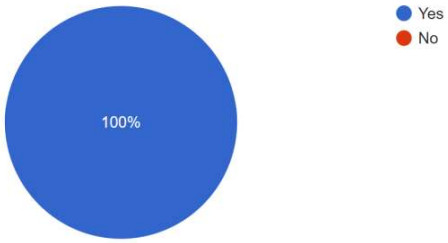
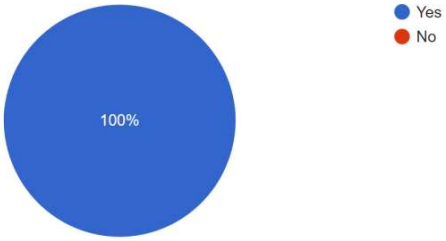
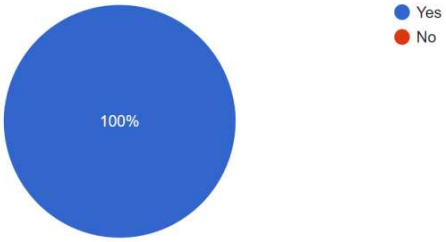
No	Pregunta	Resultados
6	Go to your user dashboard by clicking on your name at the top of the page. Then select the option Upload Data in the left panel and try to load the document sent to your email. Could you do it?	
7	Please create a new user account. Could you create it? Note: You will not be able to log in at the system with your new user account because it must requires the system administrator to activate it.	
8	What problems did you have during the use of the system?	

Tabla 3.5 Resultados de entrevistas - perfil `Administrator` (parte 1 de 4)

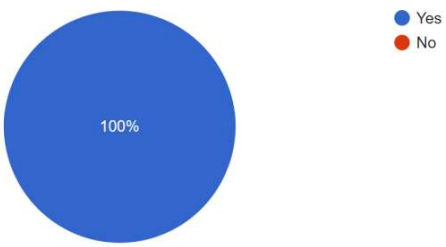
No	Pregunta	Resultados
1	Please log in at the system considering the following credentials: Email=marcos.villacis@epn.edu.ec / Password: 123LMI45678 Could you do it?	

Tabla 3.6 Resultados de entrevistas - perfil `Administrator` (parte 2 de 4)

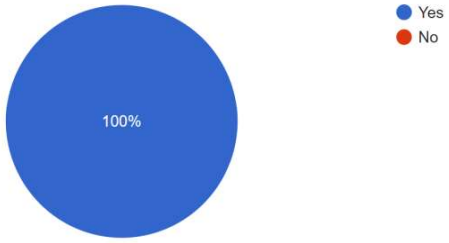
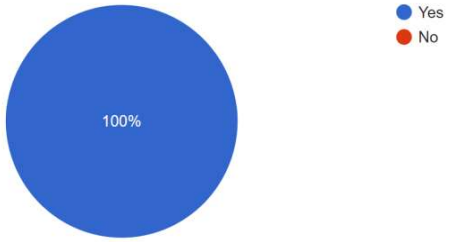
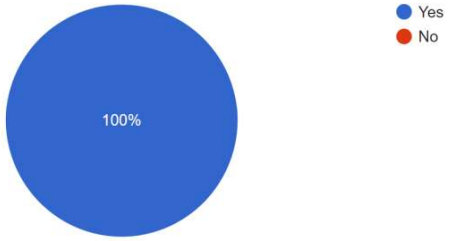
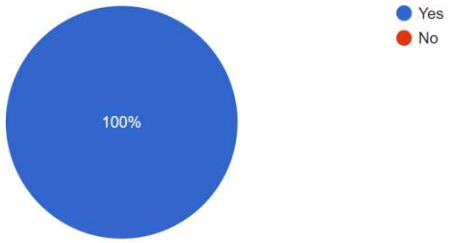
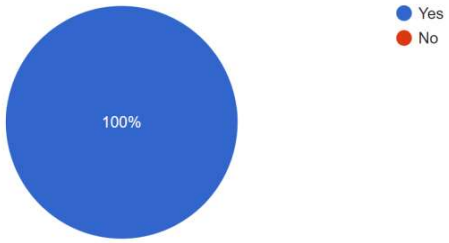
No	Pregunta	Resultados
2	Now, select the option "Search Data" and look for information considering the following parameters: Country=Ecuador / Mountain=Antisana / Glacier=Antisana-15 / Data Type=Observation / Measurement Type=Meteorological / Station=Antisana_hors_glacier / Variable=Incoming short wave radiation / Year=2007. Could you do it and visualize a temporary graph with the results?	
3	Could you also visualize information associated with the station selected?	
4	Please try to download the results. Could you do it?	
5	Now, go to the main page by clicking "LMI GREAT ICE" text at the bottom of the page. Then, in the map, select the marker called Zongo_Mass_Balance. Each marker corresponds to a station of LMI GREAT ICE. Could you visualize the documents associated with the selected station?	
6	Go to your user dashboard by clicking on your name at the top of the page. Then select the option Upload Data in the left panel and try to load the document sent to your email. Could you do it?	

Tabla 3.7 Resultados de entrevistas - perfil `Administrator` (parte 3 de 4)

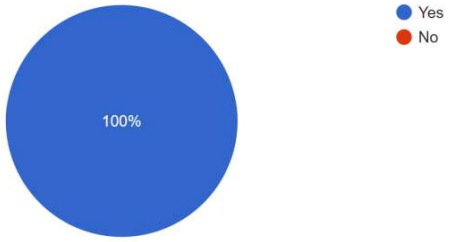
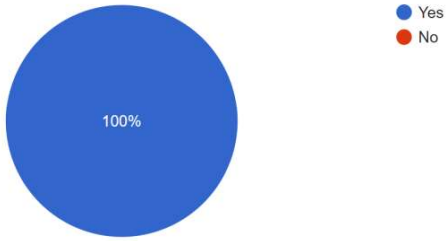
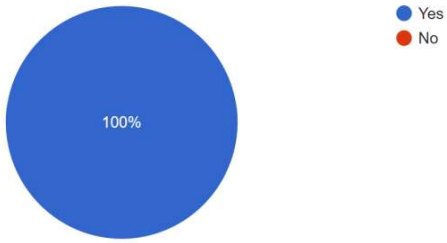
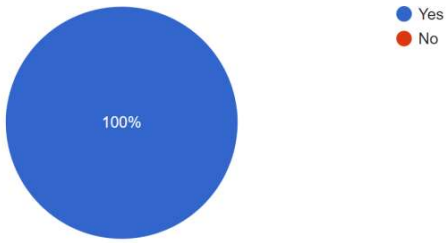
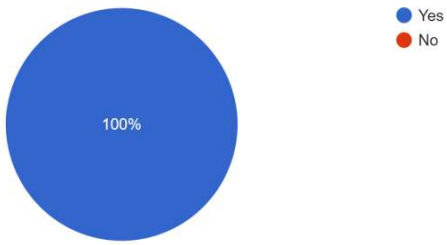
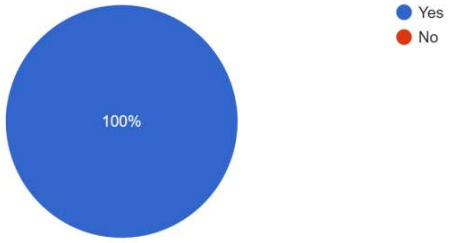
No	Pregunta	Resultados
7	Now select the option Mountains in the left panel and try to add a new register of mountain. Complete all the fields. Could you do it?	
8	Now select the option Users in the left panel. Could you visualize a list with the users registered in the system and their information?	
9	Click on the name of one of the users, you will be redirected to a new page. Try to edit the user information. Could you do it?	
10	Now, in the list of users, click on the trash icon, the value of the column "state" of that user must be set to Inactive. Did that happen?	
11	Please add a new user and set the state to Active, you must complete all the fields. Then log out of the system, and try to log in at the system with the new user account credentials. Could you do it?	

Tabla 3.8 Resultados de entrevistas - perfil `Administrator` (parte 4 de 4)

No	Pregunta	Resultados
12	Please log out of the system again. Go to the "Sign In" option and then select the "Forgot Password" button. Specify your email and click "Send". Then, verify if you received the new temporary password at your email account. Did that happen?	
13	What problems did you have during the use of the system?	I did not have problems Messages have problems

3.3 CORRECCIÓN DE ERRORES

Un resumen de los errores encontrados en el sistema y corregidos posteriormente se presenta en esta sección.

3.3.1 GRÁFICAS DE GLACIOLOGÍA

En la página principal, al realizar una búsqueda que involucrara seleccionar `Measurement Type = Glaciological`, se generaba una gráfica con los resultados. Sin embargo, esta gráfica no era correcta. Esto se debe a que para las gráficas de glaciología los resultados no se deben promediar ni acumular como es el caso de meteorología o hidrología, puesto que los valores en este tipo de medida no son recogidos cada 30 minutos o cada hora.

En glaciología, los documentos tienen 12 valores de balance de masa para cada franja altitudinal especificada, uno por cada mes del año.

Debido a esto, el gráfico resultante debe contener estos 12 valores, dependiendo de la franja altitudinal seleccionada. Por esta razón se cambió la consulta LINQ que retornaba información de la base de datos, se eliminó el elemento `DropDownList` con información

de la variable `Year` y se lo cambió por un elemento `DropwDownList` que permita retornar la lisa de documentos de glaciología para la estación seleccionada.

En la Figura 3.34 se muestran los filtros que el usuario debía seleccionar previamente. La selección mostrada producía un gráfico resultante como el que se indica en la Figura 3.35. Para generar la gráfica temporal anterior se usa el Código 3.1. Como se puede observar, la consulta LINQ involucraba seleccionar los valores de glaciología cuya estación coincidía con la seleccionada. También se pretendía hacer coincidir el año seleccionado con el año de inicio o de finalización de toma de datos del documento. Sin embargo, esto es lo que producía que la gráfica resultante fuera errónea, puesto que podía existir más de un registro de documentos en los que el año de inicio o finalización de la toma de datos coincidiera.

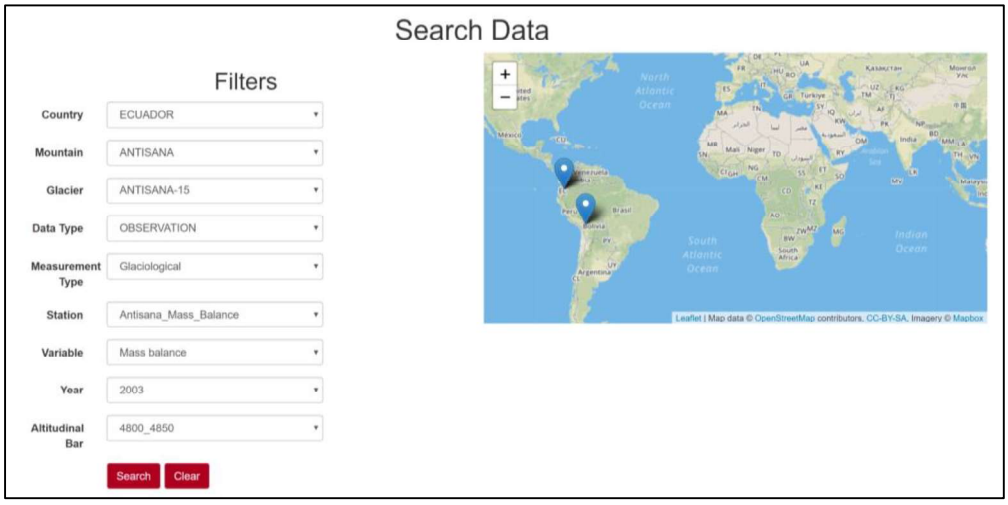


Figura 3.34 Filtros de búsqueda antes de la modificación

Figura 3.35 Gráfica de resultados antes de la modificación

```
List<glaciologicalData> searchedData = (from doc in db.enteredDatas
                                        join data in db.glaciologicalDatas on doc.idEnteredData equals data.idEnteredData
                                        where doc.idStation == u.idStation && (SqlFunctions.DatePart("year", doc.startDate) == year
                                        || SqlFunctions.DatePart("year", doc.endDate) == year) && data.altitudinalBar == u.altitudinalBar
                                        select data).ToList();
```

Código 3.1 Consulta LINQ antes de la modificación

Una vez que se detectó el error, se modificaron los filtros a seleccionar por parte del usuario en la página de búsqueda. Se reemplazó el filtro `Year` por `Document`, de esta manera, ya no se despliegan los años sino los documentos de glaciología asociados a la estación previamente seleccionada. Esto se puede observar en la Figura 3.36.

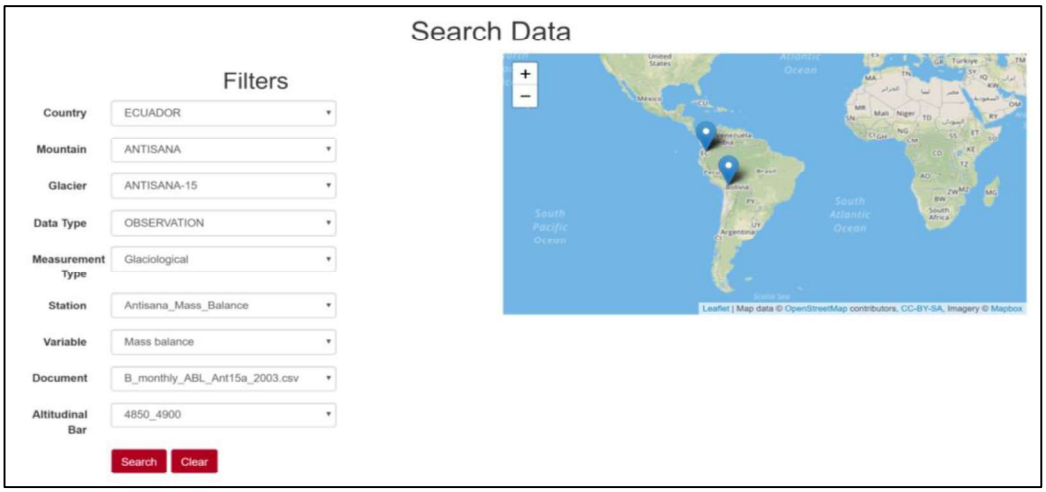


Figura 3.36 Filtros de búsqueda después de la modificación

Figura 3.37 Gráfica de resultados después de la modificación

En la Figura 3.37 se muestra la gráfica temporal resultante de los filtros seleccionados previamente. Al comparar la gráfica con los valores de la base de datos, es coincidente. En el Código 3.2 se indica la consulta LINQ utilizada para generar esta gráfica. Se puede observar que, para retornar información, se toman en cuenta únicamente el `id` del documento seleccionado en la lista desplegable y la franja altitudinal seleccionada. Así, el sistema graficará todos los valores válidos para dicha franja altitudinal del documento respectivo.

```
List<glaciologicalData> searchedData = (from doc in db.enteredDatas
                                        join data in db.glaciologicalDatas on doc.idEnteredData equals data.idEnteredData
                                        where doc.idEnteredData == u.idEnteredData && data.altitudinalBar == u.altitudinalBar
                                        select data).ToList();
```

Código 3.2 Consulta LINQ después de la modificación

3.3.2 EDICIÓN DE REGISTROS DE GLACIARES

Se observó que, al editar los registros de glaciares, no existía una validación que impida ingresar el nombre de un glaciar que ya existe previamente. Debido a esto, se podía poner el mismo nombre de otro glaciar que ya estaba almacenado en la base de datos.

En la Figura 3.38 se muestra lo que sucedía. Se modificó el registro del glaciar `ZONGO` y se cambió el nombre por `ANTISANA-15`. Como se puede observar, el registro fue actualizado correctamente sin considerar el hecho de que ya existía un glaciar con el nombre `ANTISANA-15`.

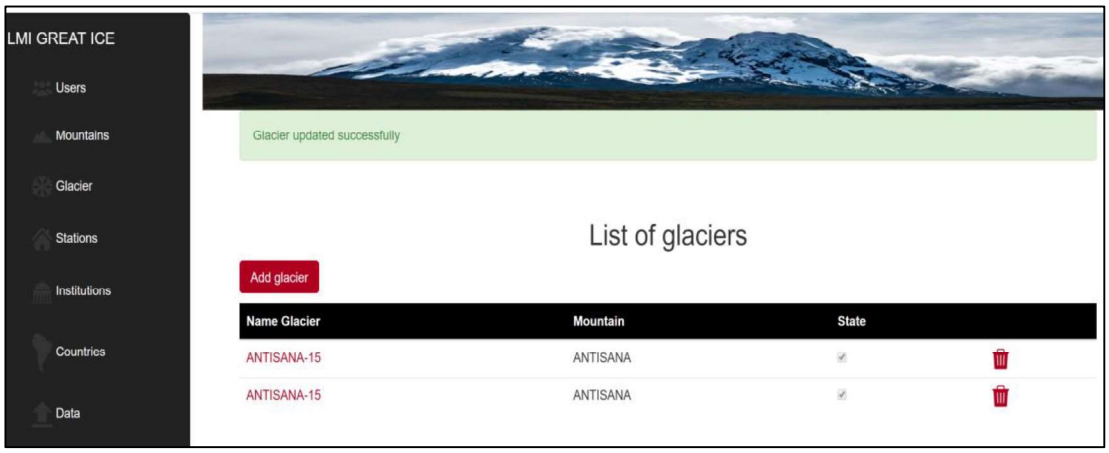


Figura 3.38 Edición de los registros de glaciares sin validación

Para controlar la existencia de registros con el mismo nombre, se añadió una porción de código al método `Edit` implementado con `POST`, del Controlador `glaciersController`.

El método `Edit` junto con la validación implementada se muestran en el Código 3.3. Como se puede observar, se colocó un condicional que compara el nombre del registro de glaciar guardado en la base de datos con el nuevo nombre del glaciar especificado por el usuario. Lo mismo se hace para el `idGlacier`. Se realiza una consulta que retorne un objeto `glacier` con las condiciones especificadas, si el objeto es nulo, quiere decir que no existe un registro de glaciar con ese nombre en la base de datos, por lo que se procede a actualizarlo. Caso contrario, se genera un mensaje indicando que el nombre de glaciar especificado ya existe.

```
94      public ActionResult Edit(int id, FormCollection collection, glacier glaciar)
95      {
96          try
97          {
98              glacier glacier = db.glaciers.Single(i => i.idGlacier == id);
99              if (db.glaciers.Where(i => i.nameGlacier == glaciar.nameGlacier && i.idGlacier != glaciar.idGlacier
100             && i.stateR == true).FirstOrDefault() == null)
101             {
102                 UpdateModel(glacier);
103                 db.SaveChanges();
104                 Session.Add("message", "Glacier updated successfully");
105                 Session["error"] = null;
106                 return RedirectToAction("Index");
107             }
108             else{
109                 Session.Add("error", "A glacier with the same name alreaddy exists!");
110                 Session["message"] = null;
111                 return RedirectToAction("Edit", id);
112             }
113         }
114         catch
115         {
116             Session.Add("error", "Error!");
117             return View();
118         }
119     }
```

Código 3.3 Método `Edit` del Controlador `glaciersController`

3.3.3 ELIMINACIÓN DE SESIONES

Los mensajes de alerta que se muestran en la aplicación web son generados a partir de cadenas de texto que se guardaron en contenedores `Session`. Sin embargo, existían casos en los que se mostraban los mensajes de varios contenedores `Session` simultáneamente. Esto se debía a que no se eliminaban los valores de dichos contenedores en los métodos respectivos. Un ejemplo del problema explicado se muestra en la Figura 3.39, donde se pretendió crear un nuevo registro de glaciar, y se muestran tanto el mensaje de error como la alerta de creación satisfactoria simultáneamente.

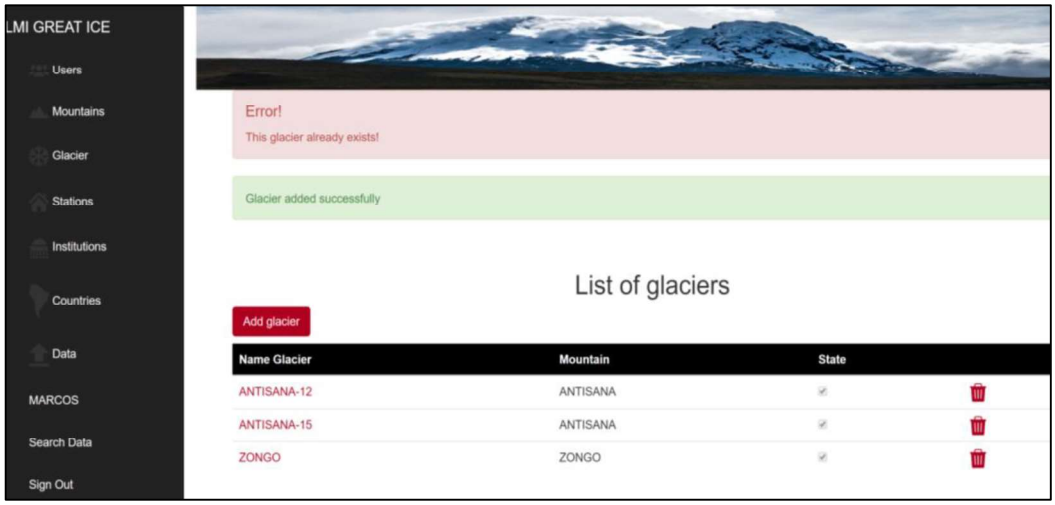


Figura 3.39 Problema generado con los mensajes de los contenedores `Session`

Para corregir el inconveniente, se procedió a poner los valores de los contenedores `Session` en `null` cuando así era necesario. En el sistema hay 3 tipos de alertas: `error`, `message` y `notification`. Entonces, si a la alerta `error` se le añade una cadena de texto mencionando el error, las otras dos alertas deben permanecer en `null` para evitar que se muestren al mismo tiempo todos los mensajes. Esto se puede observar en el Código 3.4.

```
0 referencias | 0 solicitudes | 0 excepciones
public ActionResult Create(FormCollection collection, glacier glacier)
{
    try
    {
        var newGlacier = db.glaciers.Where(a => a.nameGlacier.Equals(glacier.nameGlacier)).FirstOrDefault();
        if (newGlacier == null)
        {
            db.glaciers.Add(glacier);
            db.SaveChanges();
            //Session["error"] = null;
            Session.Add("message", "Glacier added successfully");
            Session["error"] = null;
            Session["notification"] = null;
            return RedirectToAction("Index");
        }
        else
        {
            Session.Add("error", "This glacier already exists!");
            Session["message"] = null;
            Session["notification"] = null;
            // Session["message"] = null;
            return RedirectToAction("Create");
        }
    }
    catch
    {
        return View();
    }
}
```

Código 3.4 Eliminación de mensajes en los contenedores `Session`

El código anterior logra resolver el inconveniente y causa que se muestre únicamente el mensaje correspondiente a la acción que se ejecutó, como es lo correcto. En la Figura 3.40 se muestra el error corregido.

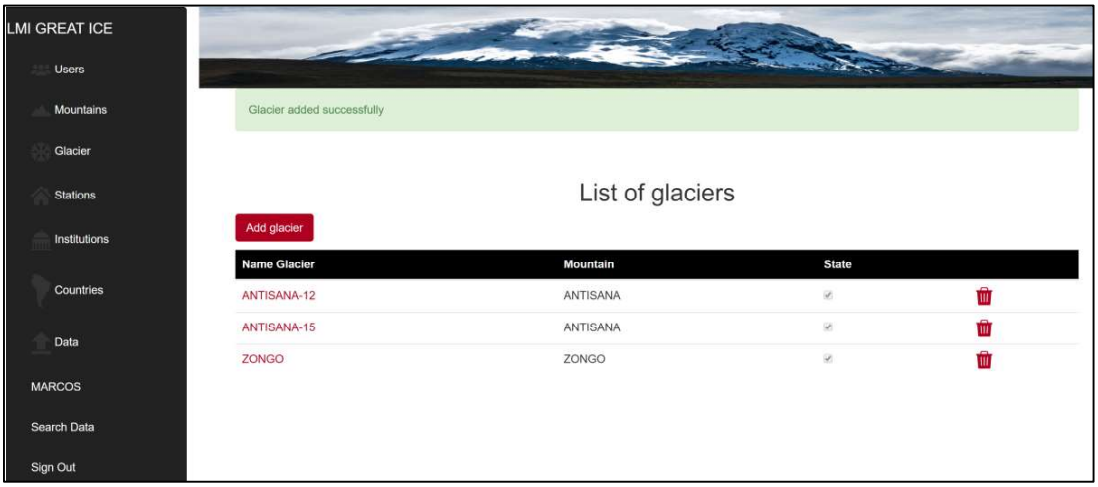


Figura 3.40 Error con sesiones corregido

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- Se desarrolló un sistema de gestión de datos para el monitoreo integral de glaciares. El sistema dispone de una base de datos y una aplicación web. La base de datos almacena la información sobre glaciares que ha sido recolectada por el proyecto de investigación LMI GREAT ICE. Por su parte, la aplicación web permite la gestión de dicha información, y cuenta con otros elementos para su visualización en Internet.
- El sistema permite realizar búsquedas de información de meteorología, glaciología e hidrología sobre los glaciares que son objeto de estudio del proyecto de investigación LMI GREAT ICE. Los resultados de las búsquedas son representados mediante gráficas temporales y pueden ser descargados en formato CSV.
- Se tiene la opción de gestionar distintos elementos en el sistema, como glaciares, montañas, usuarios, estaciones, países e instituciones.
- Los errores que se generan cuando el usuario ingresa datos con formato inválido, fueron controlados mediante el uso de *Data Annotations*.
- Se empleó la sintaxis RAZOR para combinar código HTML con C# dentro de las Vistas, permitiendo hacer uso de los *HTML Helpers* para implementar controles HTML.
- Se hizo uso de la metodología Kanban para definir tareas que se agruparon en estados de flujo de trabajo e iteraron a través de ellos hasta ser finalizadas, convirtiéndose en entregables de pequeño tamaño que juntos componen el producto final demostrable.
- Para el establecimiento de requerimientos funcionales y no funcionales, se analizaron los resultados de las entrevistas aplicadas a miembros del proyecto de investigación LMI GREAT ICE. También, se mantuvieron reuniones periódicas que permitieron añadir nuevas funciones no contempladas en las entrevistas. Finalmente, se examinó el funcionamiento de la plataforma *National River Flow Archive* (NRFA).
- En la etapa de diseño se contemplaron dos tipos de datos: `Observation` y `Simulation`. Los datos de tipo `Observation` están comprendidos por archivos con formato CSV que contienen información sobre los glaciares. Los datos de tipo `Simulation` consisten en archivos NetCDF multidimensionales que almacenan

simulaciones realizadas con la información recolectada. Sin embargo, los datos de tipo `Simulation` fueron considerados para desarrollarlos a futuro.

- Mediante Entity Framework – *Database First*, se creó el Modelo utilizado en la aplicación web a partir de la base de datos, eliminando la necesidad de crear clases y asociarlas a su respectiva tabla en la base de datos de forma manual.
- Se optó por utilizar tareas asíncronas para la carga de nueva información de meteorología, glaciología o hidrología al sistema. Esto se debe a que los documentos contienen gran cantidad de información y guardarla en la base de datos toma un tiempo considerable. Para cumplir con este cometido, se creó un Controlador llamado `AsynController` y se decidió utilizar la librería HangFire, que permite agregar hilos en segundo plano para que se ejecuten las tareas.
- Las tareas asíncronas permiten liberar el hilo principal en el que se está ejecutando algún proceso de la aplicación para que el usuario pueda realizar otras acciones simultáneamente.
- Se utilizaron contenedores de datos como `ViewBag` y `Session` para almacenar datos temporales y enviarlos desde un Controlador a otro, o desde un Controlador hacia una Vista.
- Los Controladores fueron implementados mediante la herramienta *Scaffolding*, por lo que se generaron plantillas de código genéricas que posteriormente fueron personalizadas en función de las necesidades del proyecto.
- Se utilizaron expresiones lambda para realizar consultas a la base de datos que implicaban el retorno de información que cumplía ciertas condiciones, disminuyendo de esta forma el código requerido para la consulta.
- En la codificación de las Vistas se utilizó HTML para proveer la estructura de las mismas. Asimismo, se usó CSS para aplicar los estilos y plantillas que permiten personalizar la apariencia de la Vista. Además, se utilizó jQuery, una librería de JavaScript, para añadir interactividad a los elementos de la Vista.
- Para la implementación del mapa interactivo se eligió la librería Leaflet, puesto que es de código abierto, su utilización es sencilla y, al usar mapas de OpenStreetMaps, no tiene costo.
- Se definieron plantillas estándar que deben cumplir los documentos para ser cargados en el sistema. De no cumplirse, no se podrá guardar la información en la base de datos,

puesto que el sistema detectará que la estructura del documento y los datos en él no son compatibles con lo definido.

- Al cargar un nuevo documento, el sistema lee dos veces el archivo. La primera lectura se la realiza para validar la estructura del documento y los datos contenidos. La segunda lectura permite ir insertando la información que contiene el documento en las tablas respectivas de la base de datos.
- Para los tipos de medida `Meteorological` e `Hydrological`, la generación de gráficos temporales se la hace a partir del cálculo de valores diarios, mismos que son obtenidos mediante la adición o el promedio de los valores tomados cada treinta minutos.
- Para el tipo de medida `Glaciological`, los gráficos temporales son generados considerando los valores obtenidos mensualmente.
- Se optó por agregar un atributo `state` a algunas entidades para permitir desactivar los registros y que estos no sean utilizables, pero sin eliminarlos de la base de datos. Sin embargo, los registros de los documentos y sus valores si son eliminados de la base de datos, debido a que un usuario puede querer cargar un documento que previamente ha sido cargado en el sistema, pero que tenía errores, y, dado que no se permite la modificación de esos registros en línea, deben ser completamente eliminados antes de cargar los valores correctos.

4.2 RECOMENDACIONES

- A futuro, se puede personalizar el mapa implementado para añadir interactividad o funciones extra que mejoren el aspecto visual del mismo.
- En el sistema desarrollado se usó la clase `Chart` para la generación de gráficas temporales. Sin embargo, podrían utilizarse otras librerías, como DevExtreme ASP.NET MVC Charts, que permitan una mayor personalización del gráfico agregando inclusive controles interactivos.
- Dado que el tipo de dato `Simulation` no tiene funcionalidad actualmente, se recomienda continuar con el desarrollo del sistema para que se permita agregar y manipular información de simulaciones, tales como archivos NetCDF.

- La colocación de las *Data Annotations* en las clases del Modelo, es preferible hacerlo cuando la aplicación web esté prácticamente terminada y, sobre todo, la estructura de la base de datos no tenga ningún cambio. Esto se debe a que, si se añade alguna tabla, muchas veces no servirá actualizar el Modelo desde Visual Studio, por lo que se deberá eliminar el Modelo creado por Entity Framework y crearlo nuevamente, lo que causará que las *Data Annotations* se eliminen y deban ser insertadas otra vez.
- Se recomienda continuar con el desarrollo del sistema y la mejora del aspecto visual del mismo mediante la utilización de características de CSS que no fueron consideradas en este proyecto.
- En lugar de utilizar la librería HangFire en las tareas asíncronas, se pueden implementar manualmente hilos y comparar su eficiencia con respecto al uso de la librería antes mencionada.
- Para el despliegue del sistema desarrollado podrían utilizarse los servidores de la Escuela Politécnica Nacional en vez de optar por proveedores de servicios como Microsoft Azure, debido a que el proyecto de investigación LMI GREAT ICE es una iniciativa conjunta entre varias instituciones, incluida la Escuela Politécnica Nacional.
- En un futuro, pueden añadirse nuevos criterios de búsqueda de información. De igual manera pueden modificarse los filtros o implementar un algoritmo de búsqueda que genere mayor eficiencia.
- Se recomienda implementar HTTPS para que los datos que son transferidos entre el cliente y el servidor estén cifrados.
- La base de datos contiene algunas tablas que bien podrían ser reemplazadas por listas de objetos en el código de la aplicación web, puesto que su implementación no afectaría el funcionamiento del sistema.
- Podrían implementarse mejoras en las validaciones de la estructura de los documentos. Así como también, optimizar la eficiencia del código que permite realizar estas validaciones e insertar la información en la base de datos.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] "What is REST? | Codecademy." [Online]. Available: https://www.codecademy.com/articles/what-is-rest. [Accessed: 23-Mar-2019].

[2] "Representational State Transfer (REST)." [Online]. Available: https://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html. [Accessed: 23-Mar-2019].

[3] "What is REST – Learn to create timeless RESTful APIs." [Online]. Available: https://restfulapi.net/. [Accessed: 23-Mar-2019].

[4] "Mastering REST Architecture — REST Architecture Details." [Online]. Available: https://medium.com/@ahmetozlu93/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc. [Accessed: 23-Mar-2019].

[5] R. N. Marset, "RESTRESTvsvsWeb ServicesWeb Services," pp. 1–19, 2007.

[6] "JSON." [Online]. Available: https://www.json.org/. [Accessed: 23-Mar-2019].

[7] "JSON Data Types – REST API Tutorial." [Online]. Available: https://restfulapi.net/json-data-types/. [Accessed: 23-Mar-2019].

[8] J. Galloway and D. (Computer software developer) Matson, "Professional ASP. NET MVC 5," *Prof. Asp.Net Mvc 5*, p. 622, 2014.

[9] J. A. Briones, "Automatización del proceso de aprobación de planes de trabajo de titulación en la carrera de Ingeniería Electrónica y Redes de Información," Escuela Politécnica Nacional, 2017.

[10] "Part 6: Using Data Annotations for Model Validation | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-6. [Accessed: 07-Mar-2019].

[11] "Razor syntax reference for ASP.NET | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-2.2. [Accessed: 06-Mar-2019].

[12] "ASP.NET Razor Markup." [Online]. Available: https://www.w3schools.com/asp/razor_intro.asp. [Accessed: 06-Mar-2019].

[13] "ASP.NET MVC Helpers." [Online]. Available:

https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_helpers.htm. [Accessed: 06-Mar-2019].

[14] “Bootstrap Get Started.” [Online]. Available: https://www.w3schools.com/bootstrap/bootstrap_get_started.asp. [Accessed: 14-Mar-2019].

[15] “jQuery Introduction.” [Online]. Available: https://www.w3schools.com/jquery/jquery_intro.asp. [Accessed: 14-Mar-2019].

[16] D. Cabrera, “AUTOMATIZACIÓN DE LA RECOLECCIÓN DE DOCUMENTOS DE LOS PROFESORES DEL DETRI PARA ACREDITACIÓN DE CARRERAS,” Escuela Politécnica Nacional, 2018.

[17] “ViewBag in ASP.NET MVC.” [Online]. Available: https://www.tutorialsteacher.com/mvc/viewbag-in-asp.net-mvc. [Accessed: 18-Mar-2019].

[18] “ViewData in ASP.NET MVC.” [Online]. Available: https://www.tutorialsteacher.com/mvc/viewdata-in-asp.net-mvc. [Accessed: 18-Mar-2019].

[19] “TempData in ASP.NET MVC.” [Online]. Available: https://www.tutorialsteacher.com/mvc/tempdata-in-asp.net-mvc. [Accessed: 18-Mar-2019].

[20] “Using ASP.NET MVC TempData and Session to pass values across Requests | DotNetCurry.” [Online]. Available: https://www.dotnetcurry.com/aspnet-mvc/1074/aspnet-mvc-pass-values-temp-data-session-request. [Accessed: 18-Mar-2019].

[21] “What is Internet Information Services (IIS)? - Definition from WhatIs.com.” [Online]. Available: https://searchwindowsserver.techtarget.com/definition/IIS. [Accessed: 07-Jan-2019].

[22] “IIS Web Server: (Internet Information Services).” [Online]. Available: https://stackify.com/iis-web-server/. [Accessed: 07-Jan-2019].

[23] “c# - What is the purpose of global.asax in asp.net - Stack Overflow.” [Online]. Available: https://stackoverflow.com/questions/2340572/what-is-the-purpose-of-global-asax-in-asp-net. [Accessed: 14-Mar-2019].

[24] Margaret Rouse, “What is database (DB)? - TechTarget Search SQL Server.” [Online]. Available: https://searchsqlserver.techtarget.com/definition/database. [Accessed: 17-Dec-2018].

[25] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 6th ed. Boston: Addison-Wesley, 2011.

[26] C. Coronel and S. Morris, *Database systems: design, implementation, & management*, 12th ed. Boston: Cengage Learning, 2017.

[27] José Manuel Alarcón, “¿Qué es un ORM? | campusMVP.es,” 2018. [Online]. Available: https://www.campusmvp.es/recursos/post/que-es-un-orm.aspx. [Accessed: 27-Dec-2018].

[28] L. E. Asanza, “DESARROLLO DE UN SISTEMA DISTRIBUIDO BASADO EN SOA Y MVC PARA DESPLIEGUE DE INFORMACIÓN MÉDICA DE PACIENTES,” Escuela Politécnica Nacional, 2017.

[29] José Manuel Alarcón, “Entity Framework: Code First, Database First y Model First ¿En qué consiste cada uno? | campusMVP.es,” 2018. [Online]. Available: https://www.campusmvp.es/recursos/post/entity-framework-code-first-database-first-y-model-first-en-que-consiste-cada-uno.aspx. [Accessed: 28-Dec-2018].

[30] “What is Entity Framework?” [Online]. Available: http://www.entityframeworktutorial.net/what-is-entityframework.aspx. [Accessed: 02-Jan-2019].

[31] “Entity Data Model | Microsoft Docs,” 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/entity-data-model. [Accessed: 02-Jan-2019].

[32] “Language-Integrated Query (LINQ) (C#) | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/. [Accessed: 17-Mar-2019].

[33] “Series Temporales Introducción.”

[34] T. Secular, V. Ciclicas, V. Estacionales, and A. Exponencial, “SERIES TEMPORALES (esquemas),” pp. 1–8.

[35] “Diario Panorama,” 2019. [Online]. Available: https://www.diariopanorama.com/clima/pronostico. [Accessed: 09-Jan-2019].

[36] “Leaflet - a JavaScript library for interactive maps.” [Online]. Available: https://leafletjs.com/. [Accessed: 23-Mar-2019].

[37] “Add mapping to your application with Leaflet.js • Jerrie Pelser’s Blog.” [Online]. Available: https://www.jerriepelser.com/blog/2017-12-04-add-mapping-with-leaflet/. [Accessed: 23-Mar-2019].

[38] “Leafletjs interactive map and clustering with ASP.NET MVC 5 | Tajuddin’s Blog.” [Online]. Available: https://tajuddin.chittagong-it.com/leaflet-interactive-map-and-clustering-with-asp-net-mvc-5/. [Accessed: 23-Mar-2019].

[39] “GeoJSON.” [Online]. Available: http://geojson.org/. [Accessed: 07-Jan-2019].

[40] A. Doyle, S. Gillies, S. Hagen, and T. Schaub, “RFC 7946 - The GeoJSON Format,” *IETF*, pp. 1–28, 2016.

[41] L. Angulo, “Desarrollo de un prototipo de sistema distribuido para alquiler de artículos,” p. 133, 2019.

[42] D. Anderson and A. Carmichael, *Essential Kanban Condensed*, 1st ed., vol. 39, no. 5. Seattle: Lean Kanban University Press, 2016.

[43] G. S. Matharu, H. Singh, A. Mishra, and P. Upadhyay, “Empirical Study of Agile Software Development Methodologies : A Comparative Analysis,” vol. 40, no. 1, pp. 1–6, 2015.

[44] “Maximize Your Time, Improve Efficiency with the Kanban… | LeanKit.” [Online]. Available: https://leankit.com/learn/kanban/kanban-system/. [Accessed: 28-Jan-2019].

[45] “NRFA Station Mean Flow Data for 1001 - Wick at Tarroul.” [Online]. Available: https://nrfa.ceh.ac.uk/data/station/meanflow/1001. [Accessed: 23-Mar-2019].

[46] SHAHRUKH KHAN, “List of all countries name with ISD and ISO code in SQL and CSV format - thesoftwareguy.in,” 2016. [Online]. Available: https://www.thesoftwareguy.in/list-all-countries-name-with-isd-iso-code-sql-csv-format/. [Accessed: 21-Aug-2018].

[47] “Hangfire – Background jobs and workers for .NET and .NET Core.” [Online]. Available: https://www.hangfire.io/. [Accessed: 24-Mar-2019].

[48] “¿Qué es LINQ? - TuProgramacion.com.” [Online]. Available: http://www.tuprogramacion.com/glosario/que-es-linq/. [Accessed: 16-Mar-2019].

[49] “Generación de código y plantillas de texto T4 - Visual Studio | Microsoft Docs,” 2016. [Online]. Available: https://docs.microsoft.com/es-es/visualstudio/modeling/code-generation-and-t4-text-templates?view=vs-2017. [Accessed: 02-Jan-2019].

ANEXOS

ANEXO A. Entrevistas para el levantamiento de requerimientos

ANEXO B. Plantillas definidas para cargar nuevos documentos al sistema

ANEXO C. *Scripts* para la creación y población de la base de datos

ANEXO D. Código fuente de la aplicación web

ANEXO E. Entrevistas de validación

ANEXO F. Manual de instalación y uso

Los anexos se han incluido en un CD adjunto a este documento

ORDEN DE EMPASTADO